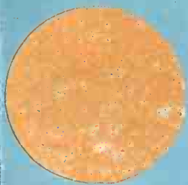


Sorin ZOICAN
Eduard POPOVICI
Dragoş CONSTANTINESCU
Adrian CONSTANTIN

ARHITECTURA SISTEMELOR DE CALCUL

ÎNDRUMAR DE LABORATOR

BUCUREŞTI 2000

Sorin ZOICAN
Eduard POPOVICI
Dragoş CONSTANTINESCU
Adrian CONSTANTIN

ARHITECTURA SISTEMELOR DE CALCUL

ÎNDRUMAR DE LABORATOR





BUCUREŞTI 2000

Introducere

Lucrarea "ARHITECTURA SISTEMELOR DE CALCUL - ÎNDRUMAR DE LABORATOR" se adresează studenților anului IV, specializarea TELECOMUNICAȚII, din cadrul Facultății de Electronică și Telecomunicații , U.P. București .

Lucrarea este structurată în 6 lucrări de laborator și o anexă.

Lucrarea 1 , " Controlul resurselor uni sistem de calcul prin intermediul sistemului de operare și accesul la resursele sistemului prin intermediul limbajelor de nivel înalt" descrie modul în care se pot controla resursele unui sietem de calcul (tastatură, imprimantă, întreruperi).

Lucrarea 2 , " Organizarea memoriei în calculatoarele personale (PC). Funcții de management al memoriei", exemplifică modul în care este organizată memoria în calculatoarele PC (memorie EMS, XMS) și modul în care memoria poate fi gestionată prin funcții C. Totodată lucrarea prezintă conceptul de program rezident în memorie.

Lucrarea 3, " Sisteme multiprocesor. Conectarea și sincronizarea procesoarelor sistemului multiprocesor", prezintă principii de realizare a sistemelor multiprocesor (conectarea cu memorie comună, principiul bus împărțirii bus-urilor). Se exemplifică aceste principii cu un sistem multiprocesor realizat cu I80x86 și ADSP2105.

Lucrarea 4, " Organizarea memoriei secundare (virtuale) în PC. Funcții de management al directoarelor și fișierelor" prezintă tipuri de memorie externă (hard disk, floppy disk) și principalele funcții pentru gestiunea acestora.

Lucrarea 5 , " Memoria cache. Gestiunea memoriei cache" ilustrează metode de mapare a memorie cahe în memoria principală a calculatorului și evaluează , cu ajutorul unui program de simulare, performanțele metodelor de mapare.

Lucrarea 6, " Moduri de lucru cu monitoarele video ale calculatoarelor personale" prezintă modurile de lucru ale display-ului calculatorului PC (text și grafic) și principalele fucții C pentru afișarea informațiilor pe ecran.

Anexa prezintă cîteva tipuri de probleme de examen.

Sugestiile și corecturile aduse de cititorii acestei lucrări vor fi întotdeauna bine venite.

București , mai 2000   Autorii

Cuprins

Controlul resurselor unui sistem de calcul prin intermediul sistemului de operare si accesul la resursle sistemului prin intermediul limbajelor de nivel inalt

Scopul lucrării

a) Studiul modalitãtilor de control prin program a resurselor unui sistem de calcul : tastaturã, display, disk. Exemple.

b) Studiul posibilitãtilor de lucru cu porturile de intrare - iesire si utilizarea sistemului de întreruperi prin intermediul limbajelor de programare de nivel înalt. Exemple.

c) Studiul unor elemente de sistem de operare (functii DOS).Exemple.

1.1. Introducere

"Arhitectura sistemelor de calcul" isi propune analizarea modului in care tehnologiile si interconectarea componentelor unui sistem de calcul influenteaza performantele acestuia. Aceasta analizã se va face din punct de vedere arhitectural fara a intra in detaliile specifice fiecãrei implementarii in parte.

Resursele unui sistem de calcul constau în : unitatea centralã de prelucrare (CPU), sistemul de memorie (memorie principalã, memorie de masã - disk, bandã magnetice) si interfete de intrare - iesire (pentru introducerea si afisarea datelor). Toate aceste resurse concurã la îndeplinirea sarcinii sistemului de calcul : efectuarea volumului de calcule asociate cu o anumitã aplicatie.

Resursele sistemului de calcul trebuie gestionate în asa fel încît să se obtină performantele maxime din punctul de vedere al vitezei de prelucrare, necesitătilor de memorare, etc.

Gestiunea resurselor sistemului de calcul revine unui program de supervizare, denumit sistem de operare. Din punct de vedere arhitectural sistemul de operare este o componenta a arhitecturii intregului sistem de calcul ce controleaza nivelul "hardware" si ofera servicii pentru programele aplicatie (ale utilizatorului).

PROGRAM APLICATIE
SISTEM DE OPERARE
ROM – BIOS
NIVEL HARDWARE

Sistemul de operare constituie un mediu pentru executia programelor de aplicatie.

Functiile sistemului de operare sînt :

- controlul executiei programelor (încărcare, lansare, terminare)
- asigurarea interfetei cu operatorul (independenta programelor relativ la hardware-ul de intrare-iesire)
- interfată cu operatorul uman
- tratare erori
- gestionarea resurselor sistemului
    - alocarea hardware a CPU, memoriei, porturilor I/O
    - protectia resurselor împotriva accesului neautorizat
    - evidenta utilizării resurselor

Componentele principale ale sistemului de operare sînt :

- **nucleul (kernel)** - alocă resursele,asigura protectia, controlează interfetele de intrare-iesire (I/O) la nivel fizic, tratează întreruperile,
- **executivul** - controlează interfetele I/O la nivél logic, gestionează sistemul de fisiere, planifică activitătile din sistemul de calcul, coordonează programele de aplicatie
- **supervizorul** - asigură interfata cu operatorul uman, asigură legătura cu celelalte nivele ale sistemului de operare, contabilizează utilizarea resurselor.

Nucleul unui sistem de operare contine o colectie de functii care sunt necesare majoritatii programelor care lucreaza pe respectivul sistem. Utilizatorul atunci cand doreste sa scrie pe hard-disc in loc sa dea comenzi direct controlerului de hard disc apeleaza o functie speciala care se gaseste in nucleu numita **apel sistem**.

Din punct de vedere arhitectural sistemele de operare se pot clasifica in: sisteme monolitice, sisteme stratificate pe mai multe niveluri, masini virtuale, sisteme client-server. In functie de arhitectura aleasa anumite functii ale sistemului de operare pot fi plasate in interiorul nucleului sau in procesele aplicatie.

Sistemul de operare DOS, cel analizat in cazul de fata este un sistem de operare monolitic alcatuit dintr-o colectie de proceduri. Fiecare procedura poate sa apeleze oricare alta procedura cand doreste acest lucru.

Sistemul de operare DOS este alcatuit din urmatoarele componente:

- Inregistrarea de boot (boot record)
- Interfata ROM BIOS (IO.SYS)
- DOS – Kernel (nucleul DOS) este alcatuit din fisierul MSDOS.SYS

- Interpretorul de comenzi standard – alcatuit din fisiserul COMMAND.COM

Inregistrarea de boot (Boot Record)

Incepe cu pista 0, fata 0, sectorul 1 al fiecarei dischete sistem DOS iar la hard-discuri se gaseste pe primul sector al partitiei DOS (bootabile). Ea genereaza mesaje de eroare daca nu se gaseste sau daca discheta afla in unitatea floppy nu este discheta sistem. Intotdeauna DOS foloseste un singur sector pentru inregistarea de boot.

Interfata ROM – BIOS

Este continuta in fisierul IO.SYS care asigura interfata cu ROM BIOS-ul placii de baza. Ea contine drivere de nivel jos (low level) pentru dispozitivele de intrare-iesire (tastatura si display (CON), interfata seriala (AUX), imprimanta (PRN),ceasul de timp real (CLOCK)), drivere pentru discurile flexibile si pentru discul hard. Accesul la aceste drivere va implica proceduri continute in memoria ROM a placii de baza. Practic in IO.SYS se gasesc mai multe tipuri de drivere care vor fi incarcate functie de tipul de memorie ROM care se gaseste pe placa de baza.
Componenta ROM – BIOS permite ca sisteme hardware diferite sa poata rula acelasi sistem de operare si aceleasi programe de tip aplicatie indiferent de componentele lor hardware. Ea actioneaza ca o frontiera de compatibilitate de aici incolo programele aplicatie nu vor tine seama de diferentele hardware.

Nucleul DOS

Este continut in fisierul MSDOS.SYS. El asigura interfata de nivel inalt intre programele aplicatie. El este constituit dintr-o colectie de functii de gestionare a fisierelor, rutine de blocare, deblocare a datelor pentru lucrul cu discul, precum si o serie de functii gata construite pentru a putea fi utilizate in programele utilizator. Aceste functii opereaza in mod independent de hardware si sunt accesibile prin intermediul intreruperi 21H).

Interpretorul de comenzi

Interpretorul de comenzi se gaseste pe disc sub forma fisierului COMMAND.COM. Interpretorul este alcatuit din urmatoarele parti componente:

- componenta rezidenta in memorie. Ea contine subrutine de tratare a intreruperilor CTRL-Break (INT 23h), de la tastatura pentru erori critice (INT 24h), intreruperile de terminare a programelor (INT 22h). La sfarsitul oriocarui program se verifiaca daca respectivul program a rescris componenta tranzitorie a COMMAND.COM-ului daca da se va reincarca iar daca nu se poate reincarca (nu se gaseste pe respectivul disc) se va opri functionarea sistemului.m Intregul proces de tratare a erorilor in cazul sistemului de operare DOS este continut in componenta COMMAND.COM.
- componenta de initializare a sistemului de operare DOS. Ea este cea care se incarca in memorie odata su sistemul de operare DOS. Ea determina unde anume vor fi incarcate in memorie programele

utilizatorului. Cand operatia de incarcare s-a terminat ea nu mai este necesara si spatiul respectiv de memorie va fi folosit in alte scopuri.

- Componenta tranzitorie. Ea se incarca la sfarsitul memoriei si contine interpretorul de comenzi. Aceasta contine interpretoarele de comenzi externe si interpretorul de fisiere batch.
- Componenta de afisare a prompterului DOS. Ea este cea care citeste comanda de la dispozitiul standard de intrare si executa comanda.Pentru comenzi externe transfera controlul componentei tranzitorii.

Incarcarea sistemului de operare DOS

- La resetarea software a calculatorului prin intermediul comenzilor (Ctrl+Alt+Del) sau la resetul hardware se preda controlul componentei ROM BIOS de pe placa de baza si eventual componetelor ROM BIOS de pe placile de extensie. Prima portiune de program executata din memoria ROM BIOS o reprezinta testele de verificare la pornire numite POST (Power On Self Test) care verifica starea masinii si ruleaza diferite programe de inspectie(ROM Bootstarap). Dupa aceasta se trece la cautarea discului de pornire se citeste primul sector al discului de boot (Boot Sector).Acesta contine un program numit Disk Bootstrap.
- Disk Bootstrap-ul este citit in memorie si I se preda controlul. In acest moment sistemul de operare nu este incarcat in memorie si harta memoriei arat astfel: (figura .1).

00000h	Vectori de intrerupere
00400h	.....
	Disk – Bootstrap
	....
FFFFFh	ROM-Bootstrap

Fig. 1 Harta memorie dupa "Disk-Bootstrap"

Programul incarcator citeste primul sector al discului si vede daca pe acest disc se gasesc fisierele IO.SYS si MSDOS.SYS in caz ca le gaseste continua procesul de incarcare mai departe iar in caz contrar afiseaza un mesaj de eroare. In caz ca le gaseste le incarca si preda controlul fisierului IO.SYS.

- Fisierul IO.SYS contine doua componente:
  - Componenta BIOS
  - Componenta SYSINIT

Componenta SYSINIT are sarcina incarcarii sistemului de operare. In acest moment harta memoriei arata astfel (figura 2):

00000h	Vectori de intrerupere
00400h	BIOS (din IO.SYS)
	SYSINT (din IO.SYS)
	DOS-kernel (din MSDOS.SYS)
	Disk-Bootstrap
FFFFFh	ROM

Fig 2. Harta memoriei dupa incarcarea modulelor IO.SYS si MSDOS.SYS

Componenta SYSINIT determina starea echipamentelor periferice, initializeaza unitatile atasate,seteaza parametrii sistem si incarca toate driverele instalate conform fisierului CONFIG.SYS, seteaza vectorii de intrerupere.

- Se preda controlul portiunii de intializare a modului DOS-Kernel. Acesta initializeaza tabelele sale interne si initializeaza vectorii de intrerupere pentru intreruperile 20h –2Fh. In acest moment sistemul de operare DOS este functional
- Modulul SYSINIT apeleaza functia EXECpentru incarcarea interpretorului de comenzi si il lanseaza pe acesta in

executie.Componenta tranzitorie a COMMAND.COM-ului este suprascriasa peste SYSINIT.

- Interpretorul de comenzi afiseaza prompteru si asteapta ca utilizatorul sa introduca comenzi. Harta memoriei arata ca in (figura 4).

00000 h	Vectori de intrerupere
00400h	DOS-kernel (din MSDOS.SYS)
	Drivere
	Partea rezidenta din COMMAND.COM
	Zona de procese tranzitorii
	SYSINIT
FFFFFh	ROM

Fig. 4 Harta memoriei dupa incarcarea S.O.

In cazul sistemului DOS apelurile sistem nu se fac prin intermediul unei functii cum este cazul Unix-ului ci prin intermediul unei intreruperi ("software") care plaseaza anumite date in registrii microprocesorului si apoi apeleaza o intrerupere. Viteza sistemului de operare DOS provine in parte si din acest tip de apel sistem.

Intreruperea – permite calculatorarelor sa trateze sarcinile neprevazute ce apar in executia programelor. Atunci cand un calculator primeste o intrrupere el suspenda activitatea in curs si trateaza intruperea respectiva nu inainte de a salva contextul sarcinii anterioare.

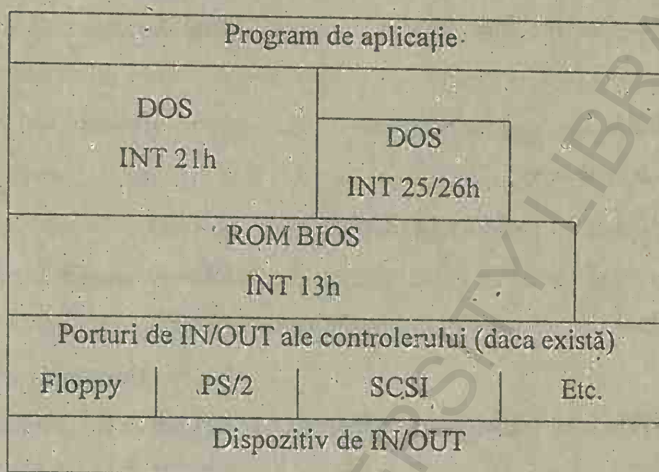
Intreruperile pentru IBM-PC pot fi clasificate in:

- dupa cei care le-au propus:

- intreruperi hard INTEL – integrate in procesor de proiectantii procesoarelor INTEL (0,1,3,4)
- intreruperi hard – definite de IBM (2,8,9,11,12,13,14,15 – supervizate de controlerul de intreruperi 8259A)
- intreruperi soft integrate in sistemul de operare D.O.S. (integrate in memoria BIOS – 5,16,28,72 – intreruperi DOS disponibile numai sub sistemul de operare DOS – 32-96)
- intreruperi ale softului de aplicatii (96 – pana la 103)
- intreruperi de adresa – utilizate pentru accesul rapid la anumite informatii (tabela vectorilor de intreruperi) (acestea sunt 29,30,31)

Sistemul de intrare – ieşire

In figura de mai jos este prezentat pe scurt sistemul de intrare-iesire in cazul sistemului de operare DOS. Programul aplicaţie care doreşte un acces la dispozitivele de IN/OUT transferă aceste date prin intermediul intreruperii DOS INT 21h. Aceasta la rândul ei transferă datele componentei BIOS prin intermediul întreruperii 13h. Aceste date sunt transformate de către BIOS în comenzi adresate direct controlerului de IN/OUT. Controlerul executa comenzile si intoarce datele care sunt transferate înapoi programului aplicaţie. Pentru ca BIOS-ul să poată comunica cu controlerul dispozitivului de IN/OUT trebuie ca programul din BIOS trebuie să se adreseze unui anumit tip de controler. De aceea se întâmplă frecvent ca în cadrul memoriei BIOS să existe mai multe tipuri de programe dedicate unui anumit dispozitiv de IN/OUT fiecare fiind specific unui anumit producător.



Independența sistemului de operare de partea hardware este relevată de existența întreruperii 13h care transformă comenzile sistemului de operare în comenzi speciale ale hardware-lui dedicat. Eventualele modificări hardware vor trebui însoțite doar de modificarea componentei BIOS a sistemului de calcul.

1.2. Functii DOS uzuale

Principalele componente ale acestui sistem de operare sînt : BDOS (sistemul de operare de bază) si BIOS (componentă ce asigură interfata cu dispozitivele I/O).

Tabelul următor ilustrează cîteva dintre functiile DOS (asociate componentei BDOS):

Nr. functie	*Specificare*	*Reg. intrare*	*Reg. iesire*
01H	citire tastatura	AH=01	AL <-- caracter citit
02H	afisare pe display	AH=02 DL <-- caracter de afisat	
05H	tiparire la imprimanta	AH=05 DL <-- caracter de tipãrit	
09H	afisare sir pe display	AH=09 DS:DX <-- adresa sirului (terminat cu '$')	
2AH	afisare datã	AH=2A AL <-- ziua saptamînii CX <-- anul DH <-- luna DL <-- ziua lunii	
2CH	afisare timp	AH=2C CH <-- ora CL <-- minutul DH <-- secunda DL <-- sutimi de secundã	

Existã de asemenea functii pentru crearea unui director (fisier), stergerea unui director (fisier), setare atribute fisiere, controlul dispozitivelor I/O, alocarea memoriei, executie si încãrcare programe, terminare programe.

În general, functiile DOS nu sînt utilizate direct ; controlul resurselor sistemului de calcul se realizează mai comod prin intermediul unor proceduri (sau functii) asociate unor limbaje de programare de nivel înalt (Turbo Pascal, Turbo C, Turbo C++ ,etc.). Functiile DOS pot fi utilizate în situatiile care necesită timpi de executie mici ; în acest caz apelul functiilor DOS se realizează în limbaj de asamblare, urmărind registrele implicate (ca în tabelul anterior).

1.3. Exemple de functii C pentru controlul resurselor sistemului de calcul

bdos : Apel functii DOS

Declaratie: int bdos(int dosfun, unsigned dosdx, unsigned dosal);

Param.	Semnificatie
dosfun	Defineste functia DOS
dosdx	Valoarea registrului DX
dosal	Valoarea registrului AL

Intoarce valoarea actualizata a registrului AX

---

bioskey : Lucrul cu tastatura utilizind in mod direct BIOS

Declaratie : int bioskey(int cmd);

cmd	Functie si valoare intoarsa
0	Intoarce tasta apasata
	Daca cei mai putin semnificativi 8 biti sint nenuli

	atunci valoarea este caracter ASCII;in caz contrar bitii mai semnificativi reprezinta codul extins al tastei
1	Testeaza daca s-a apasat o tasta; daca nu s-a apasat intoarce 0; daca s-a apasat CTRL-BRK intoarce -1; in caz contrar intoarce valoarea tastei.
2	Testeaza taste de control astfel :

Bit	*Value*	*Semnificatie*
0	0x01	Shift Dreapta tastat
1	0x02	Shift Stinga tastat
2	0x04	Ctrl tastat
3	0x08	Alt tastat
4	0x10	Scroll Lock activ
5	0x20	Num Lock activ
6	0x40	Caps activ
7	0x80	Insert activ

---

biosdisk : Utilizare disk cu ajutorul BIOS (in mod direct)

Declaratie : int biosdisk(int cmd, int drive, int head, int track, int sector,int nsects, void *buffer);

NOTA: biosdisk opereaza sub nivelul fisierelor. Se pot distruge continutul
fisierelor si al directoarelor pe hard disk !!!

Param.	Semnificatie
cmd	Indica operatia de efectuat
drive	Specifica drive-ul ce va fi utilizat
head	Specifica sectorul de start
sector	
nsects	Numar de sectoare pentru transfer (1 sector = 512 octeti)
buffer	Adresa in memorie (pentru transfer)

Valoare intoarsa:

Operatie cu succes : octetul superior = 0;

octetul inferior contine numarul de sectoare (citite, scrise, verificate)

Operatie cu eroare : octetul superior = una din valorile

Val.	Descriere
0x01	Comanda eronata
0x02	Adresa inexistenta
0x03	Incercare de scriere pe disk protejat
0x04	Sector inexistent
0x05	Eroare de reset (hard disk)
0x06	Disk-ul a fost schimbat de la ultima operatie
0x07	Eroare de drive
0x0A	Sector defect
0x0B	Pista defecta
0x0C	Pista inexistenta
[illegible]	[illegible]

0x10	Eroare de verificare la citire CRC/ECC
0x11	Eraore CRC/ECC corectata
0x20	Controler Defect
0x40	Operatie de cauatre esuata
0xAA	Disk- ul nu e gata
0xE0	Eroare de stare

---

biosprint : Tiparire la imprimanta utilizind BIOS in mod direct

Declaratie : int biosprint(int cmd, int abyte, int port);

Arg.	Semnificatie
abyte	Caracterul de tiparit;valoare intre 0 si 255.
cmd	Functia imprimantei
0	Tipareste caracterul din abyte
1	Initializeaza imprimanta
2	Citeste starea imprimantei

Daca cmd = 1 or 2 , abyte este ignorat.

port Identifica imprimanta : 0 = LPT1, 1 = LPT2, etc.

Starea imprimantei este obtinuta astfel :

Bit	Valoare	Starea imprimantei
0	0x01	Time out
3	0x08	Eroare I/O
4	0x10	Selectata
5	0x20	Lipsa hirtie

6	0x40	Acknowledge
7	0x80	Not busy

---

inport / inportb - citeste un cuvint/octet de la un port

outport / outportb - scrie un cuvint/octet la un port

Declaratii :

```
unsigned      inport  (unsigned portid);
unsigned char inportb (unsigned portid);
void          outport (unsigned portid, unsigned      value);
void          outportb(unsigned portid, unsigned char value);
```

Citirea unui cuvint se efectueaza astfel :octetul inferior de la adr , iar cel superior de la adr+2.(Portul de intrare de 8 biti,conectat pe bus-ul de date inferior)

Scrierea unui cuvint se efectueaza astfel :octetul inferior al cuvintului la adr, iar cel superior la adr+1.(Portul de iesire de 16 biti).

Argument	Semnificatie
portid	Adresa portului
value	Cuvintul / octetul citit/scris

1.4. Exemple de programe

{ASCL1.C - lucrul cu fisiere si cu grafica }

```
#define n    100
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <dos.h>
#include <graphics.h>

int gdriver = DETECT, gmode, errorcode;

void main(void)
{
FILE *fis;
int i;
double sgn[n];
double a=0.001;
char str[9];
int s,s1,s2;
double b,j;

clrscr();
fis=fopen("sgn.dat","wt");
/*deschide fisierul "sgn.dat", pentru scriere in
mod text */
for(i=1;i<n+1;i++)
    {
//calcule numerice
```

```
    j=0.01*i;
    b=1/(1-a*j*j);
    sgn[i]=(1/j)*log(b);
    sprintf(str,"%f\n",sgn[i]);
//  converteste  sgn[i]  din  float  in  sir  de
caractere
    fwrite(&str,sizeof(str),1,fis);
// scrie in fisier
    }
fclose(fis); // inchide fisierul
initgraph(&gdriver, &gmode, "c:\\bc\\bgi\\");
  s=10; //factor de scala
//afisare grafica  (mod grafic 640 x 480 pixeli)
  for(i=1;i<n-1;i++)
    {
    s1=480-100000*sgn[i];
    s2=480-100000*sgn[i+1];
    line(i*s,s1,(i+1)*s ,s2);
    }
   getch();
   closegraph();
}
```

{ASCL2.C - lucrul cu întrruperi}

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>

#define INTR 0X1C    /* Intrerupere de la timer PC
- la 50ms */
```

```
void interrupt ( *oldSCI)(void);
//rutina de servire a intreruperii anterioara

int count1=0,count=0,flag=0;

//noua rutina de servire a intreruperii
void interrupt newSCI(void)
{
   //numara 20 intrruperi - 1 sec
   count++;
   if (count==20) {count1++;flag=1;count=0;}

/* apel old SCI */
   oldSCI();
}

void main(void)
{
    clrscr();
    printf("Start asteptare intreruperi\n");

/* salveaza vectorul de intrerupere vechi*/
   oldSCI = getvect(INTR);

/* scrie noul vector se intrerupere*/
   setvect(INTR, newSCI);

/* bucla de asteptare intreruperi pina cind
counter > 20 */
   while (count1 < 20)
```

```
   if (flag==1) { printf("%d\n",count1);flag=0;}

    printf("Asteptare intreruperi incheiata\n");
    printf("Tastati orice tasta pentru a termina
programul\n");

/* reface vechiul vector de intrerupere*/
   setvect(INTR, oldSCI);
   while (!kbhit());
}
```

{ASCL3.C - functii DOS}

```
#include <stdio.h>
#include <dos.h>

/* citeste unitatea de disc curenta ca 'A', 'B',
... */
char current_drive(void)
{
   char disc;

   /* citeste discul curent ca 0, 1, ... */
   disc = bdos(0x19, 0, 0);
   return('A' + disc);
}

void main(void)
{
```

```
   printf("Unitatea de disc curenta este %c \n",
current_drive());
}
```

{ASCL4.C - lucrul cu tastatura (utilizare directa BIOS)}

```
#include <stdio.h>
#include <bios.h>
#include <ctype.h>

#define RIGHT  0x01
#define LEFT   0x02
#define CTRL   0x04
#define ALT    0x08

void main(void)
{
   int key, modifiers;

   /* functia 1 intoarce 0 pina cind se apasa o
tasta */
   while (bioskey(1) == 0);

   /* functia 0 intoarce codul tastei apasate */
   key = bioskey(0);

   /*functia 2 determina daca au fost utilizate
taste de control
   SHIFT stinga sau SHIFT dreapta, CTRL, ALT */
   modifiers = bioskey(2);
```

```
    if (modifiers)
    {
       printf("[");
       if (modifiers & RIGHT) printf("RIGHT");
       if (modifiers & LEFT)  printf("LEFT");
       if (modifiers & CTRL)  printf("CTRL");
       if (modifiers & ALT)   printf("ALT");
       printf("]");
    }
    /* afisaeza caracterul citit de la tastatura */
    //test  daca  tasat  apasata  este  caracter
alfanumeric
    if (isalnum(key & 0xFF))
       printf("'%c'\n", key);
    else
       printf("%#x\n", key);
       //afisare  in  hexazecimal  (forma  alternata,
cu prefixul 0x)
}
```

{ASCL5.C - functii BIOS - disk}

```
#include <bios.h>
#include <stdio.h>

void main(void)
{
   int result;
   char buffer[512];
```

```
   printf("Test daca drive-ul a: e pregatit
(Ready) \n");
   result = biosdisk(4,0,0,0,0,1,buffer);
   result &= 0x02;
   (result) ? (printf("Drive A: Ready\n")) :
          (printf("Drive A: Not Ready\n"));

}
```

{ASCL6.C - functii BIOS - imprimanta}

```
#include <stdio.h>
#include <conio.h>
#include <bios.h>

void main(void)
{
   #define STATUS  2      /* comanda de citire a
starii imprimantei*/
   #define PORTNUM 0      /* identificator port
pentru LPT1 */

   int status, abyte=0;

   printf("Opriti imprimanta. Apasati orice tasta
pentru a continua\n");
   getch();
   status = biosprint(STATUS, abyte, PORTNUM);
   if (status & 0x01)
      printf("Stare : time out \n");
```

```
    if (status & 0x08)
       printf("Stare : Eroare I/O \n");
    if (status & 0x10)
       printf("Stare : Imprimanta selectata \n");
    if (status & 0x20)
       printf("Stare : Lipsa hirtie \n");
    if (status & 0x40)
       printf("Stare : Acknowledge \n");
    if (status & 0x80)
       printf("Stare : Not busy \n");

}
```

{ASCL7.C - lucrul cu porturile}

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
   int port = 0;
   int value = 'C';

outport(port, value);
printf("Valoarea %d a fost scrisa la portul de
adresa %d\n", value, port);
}
```

1.5. Desfãsurarea lucrãrii

a) Sa se precizeze pozitionarea tabelului vectorilor de intrerupere, modul de organizare a unei subrutine de tratare a intreruperilor utilizand dezansamblarea unei locatii din memoria ROM a calculatorului IBM – PC.

b) Sã se studieze exemplele de la punctul 1.4. Sã se verifice corectitudinea functionãrii programelor.

c) Sã se realizeze un program ce afiseazã pe ecran codul unei taste apãsate, folosind functii C din biblioteca bdos.

d) Sa se scrie un program ce afiseaza codul ASCII al unei taste citite cu ajutorul functiilor bdos.

e) Sa se scrie un program ce afiseaza codul tastei apasate utilizand functiile bioskey().

f) Sã se realizeze un program de tipãrire la imprimantã folosind functii bdos.

g) Sã se realizeze un program de tipãrire la imprimantã folosind functia biosprint.

h) Sã se scrie un program ce implementeazã un cronometru digital care afiseazã pe ecran minutele si secundele, utilizînd sistemul de întreruperi (ceasul de timp real al PC).

i) Sã se scrie un program similar celui de la punctul e) , dar utilizînd functii DOS. (afisare an,luna, zi, ziua sãptãmînii, ore , minute, secunde).

j) Sa se scrie un program ce trimite la un port serial semnalul :

k) Sã se studieze programul demostartiv BGIDEMO.C din kit-ul de instalare Borland C ++.

Organizarea memoriei în calculatoarele personele (PC). Funcţii de managenement al memoriei .

Scopul lucrării

a) Descrierea organizării memoriei în calculatoarele PC. Definiţii ale tipurilor de memorie.
b) Funcţii C pentru managementul memoriei. Exemple.
c) Programe rezidente în memorie. Exemple.

1. Memoria în calculatoarele PC

Primele calculatare personale puteau accesa doar 64ko de memorie. Apariţia procesoarelor mai performante (Intel 8086) a permis adresarea unui spaţiu de memorie de 1Mo. Acest megaoctet fiind format din 16 blocuri de câte 64ko. Fiecare bloc având o destinaţie specială.

Segment	Destinaţie
0 – 10	Memorie utilizator de lucru
A	Memorie video utilizat pentru adaptorul de grafică extinsă EGA, VGA
B	Memorie video obişnuită. Modul alfanumeric monocrom utilizează zonele B0000-B7FFF, iar modul alfanumeric color utilizează zona B8000-BFFFF.
C	Folosit pentru extensia programelor din memoria ROM.
D	Folosit deasemenea pentru extensia progrmelor din memoria ROM.
E	Memoria sistemului ROM – BIOS
F	Memoria ROM- BIOS si ROM BASIC

In prezent astăzi sunt folosite numai următoarele locaţii de memorie.

Adresa	Utilizare
00000 – 9FFFF	Memorie RAM pentru programele de aplicaţie şi sistemul de operare
B0000 – B3FFF	Memorie pentru ecranele monocrom
B4000 – B3FFF	Memorie pentru ecranele color
CC000 – EFFFF	Memorie ROM pentru aplicaţii
F0000 – FFFFF	Memorie ROM pentru BIOS

Spaţiu de memorie, într-un calculator IBM PC (sau combatibil) care rulează sistemul de operare DOS este împarţit astfel:

- Zona de memorie de bază
- Zona de memorie superioară (UMA)
- Zona de memorie înaltă (HMA)
- Memoria expandată (EMS)
- Memoria extinsă (XMS)

Memoria de bază

Zonă de 640ko - începînd de la adresa 0 - care se numeşte *memoria DOS inferioară* (*low DOS memory*). Memoria de bază este rezervată pentru unele utilizări importante care sunt fundamentale pentru buna funcţionare a calculatorului. Acesta memorie cuprinde:

- Tabela vectorilor de întrerupere – locul unde sunt plasate rutinele de tratare a întreruperilor, un spaţiu de 1024 octeţi care cuprind spaţii pentru 256 de întreruperi între adresele 00000 – 00400.
- Spaţiu de lucru cu ROM – BIOS –ul.

şi o zonă în continuarea primei - de 384ko - denumita *memoria DOS superioară* (*high DOS memory*) sau *memorie convenţională*.

Sistemul de operare, programe pentru controlul dispozitivelor I/O (*device handlers* sau *drivers*) , programe rezidente (ce rămîn în memorie) ocupă o parte din zona memoriei DOS inferioare.

Uzual , zona de memorie DOS inferioară este constituită din circuite RAM.

Zona de memorie DOS superioară conţine 64ko de memorie ROM (ultimii 64ko) ce reprezintă sistemul de intrare / ieşire (*Basic Input Output System - BIOS*). Se mai numeşte *UMB - Upper memory Block*.

Zona de memorie DOS superioară conţine şi dispozitive de intrare / ieşire : plăci video , plăci de reţea.

Programele de aplicaţie utilizează zona de memorie DOS inferioară (programe care nu lucrează în modul de lucru protejat al procesoarelor I80286,I80386 sau I80486).

Managementul de memorie pentru calculatoarele PC trebuie să asigure două cerinţe :

- eliberarea unei zone cît mai mari din zona de memorie DOS inferioară (pentru rularea aplicaţiilor în mod *real* sub sistemul de operare DOS)

- accesul la memoria peste limita de 1Mo (pentru a exploata avantajele procesoarelor evoluate - I80286,I80386, I80486 - ce pot adresa

spaţii de adrese superioare limitei de 1Mo , atunci cînd lucrează în modul *protejat* cu păstrarea compatibilităţii cu sistemul de operare DOS.

Modul *protejat* modifică radical adresarea memoriei :se utilizează o sdresă logică formată dintr-un *selector* (care determină *adresa de bază a segmentului* - pe 32 biţi - pentru I80486 , *limita segmentului* şi *drepturile asociate segmentului* ; selectorul este un index într-un *tabel de descriptori de segmente*) şi un offset (pe 32 de biţi - pentru I80486). Adresa fizică , obţinută prin adunarea adresei de bază cu offsetul , este de 32 biţi , ceea ce permite adresarea a maxim 2^{32} octeţi = 4Go.

1.1. Memoria expandată (Expanded memory)- EMS

Memoria expandată este memoria peste limita de 1Mo , dar care apare în interiorul spaţiului de adrese asociate primului Mo. Există o metodă (specificaţie) ce face posibilă accesarea unei zone de memorie de adresă peste 1Mo în spaţiul de adrese din interiorul primului Mo. Această specificaţie se nuneşte *Expanded Memory Specification - EMS* . Calculatorul trebuie să posede un hardware special (placă de memorie EMS) ; acest hardware dedicat face ca memoria de la adrese peste 1Mo să se "vadă " din interiorul unei ferestre de memorie din spaţiul de adresare de pînă la 1Mo (figura 1).

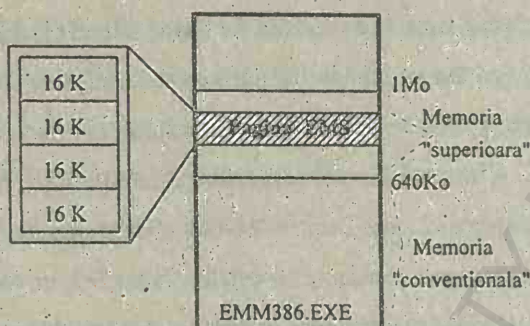


Figura 1. Memoria EMS

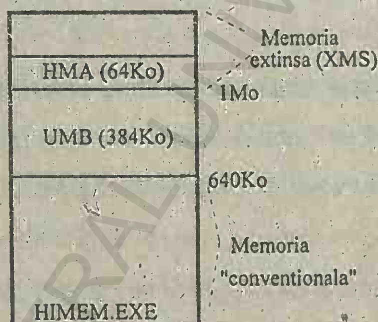


Figura 2. Memoria XMS

Un program special (manager de memorie) numit EMM386.EXE trebuie încărcat pentru a avea acces la memoria expandată. Orice procesor (I8086 - I80486) poate avea acces la memoria expandată , chiar şi în modul real de lucru.

Dezavantajul memoriei expandate este acela că utilizează un hardware complex.

1.2. Memoria extinsã (Extended memory) - XMS

Memoria extinsã este memoria peste limita de 1Mo , dar care este accesatã *direct* , atunci cînd procesorul lucreazã în modul de lucru protejat (nu mai este necesar un hardware specializat pentru a *mapa* memoria superioarã într-un spaţiu de adrese de pînã la 1Mo (figura 2).

Primii 64ko peste limita de 1Mo poartã denumirea de *High Memory Area* şi pot sã fie *accesaţi de cãtre procesor (I80286-I80486) în modul real.*

Un program special (manager de memorie) denumit HIMEM.SYS trebuie încãrcat pentru a asigura accesul al memoria extinsã (eXtended Memory Specification - XMS).

Programul EMM386.EXE poate utiliza memoria XMS pentru a emula memoria EMS (dacã nu existã placa de memorie expandatã în sistem).

Memoria EMS nu trebuie confundatã cu memoria XMS. Memoria EMS a fost dezvoltatã pentru a creşte performanţele sistemului pentru modul real , iar memoria XMS a fost conceputã pentru lucrul pe modul protejat. Memoria expandatã nu ocupã o zonã specificã de adrese ci este asociatã cu un mecanism special de mapare ce face ca aceastã memorie sã poatã fi accesatã, în pagini, din spaţiul de adrese specifice modului real (pînã la 1 Mo). Memoria extinsã apare în mod natural pentru modul de lucru protejat.

2. Funcții C pentru managementul memoriei

malloc **<ALLOC.H, STDLIB.H>**

Declaratie: void *malloc(size_t size);

Functia malloc aloca un bloc de date de dimensiune size octeti din memoria nealocata (heap). Se permite alocarea explicita de memorie (prin specificarea directa a unei dimensiuni de memorie).

Memoria heap este utilizata pentru alocarea dinamica;tot spatiul de memorie de la sfirsitul segmentului de date pina la virful stivei programului (cu exceptia unui mic spatiu utilizat de DOS plus un spatiu de rezerva) este disponibil pentru alocare dinamica.

Spatiul de memorie alocat asfel constituie parte integranta din program si nu este recunoscut ca bloc de date DOS de catre sistemul de operare.

Valoare intoarsa

- operatie cu succes : un pointer catre blocul alocat
- operatie cu eroare : pointerul NULL (daca nu exista spatiu in heap)

Daca argumentul size=0 functia malloc intoarce NULL.

free **<STDLIB.H, ALLOC.H>**

Declaratie: void free(void *block);

Functia free elibereaza blocuri de memorie alocate cu functia malloc.

Valoare intoarsa : nici una.

allocmem **<DOS.H>**

Declaratie: int allocmem(unsigned size, unsigned *segp);

Functia allocmem utilizeaza functia DOS 0x48 pentru a aloca un bloc de memorie DOS caracterizat print-un antet de 16 octeti astfel :

- octetul 0 : este 'M' sau 'Z'
- octetii 1 si 2 : reprezinta adresa de segment a blocului alocat (0000 daca blocul a fost eliberat)
- octetii 3 si 4 : reprezinta lungimea blocului in multiplii de 16 octeti
- octetii 5 - 15 : neutilizati

Semnificatia parametrilor este :

Parametru	Semnificatie
size	Numarul de grupuri de 16 biti cerute
segp	Pointer catre un segment alocat

NOTA: Functia malloc nu poate coexista cu funcţia allocmem.

Valoare intoarsa

Operatie cu succes : -1

Operatie cu eroare : lungimea blocului de dimensiune maxima (dar mai mica decit cea solicitata, si un cod de eroare DOS)

heapwalk <ALLOC.H>

Declaratie: int heapwalk(struct heapinfo *hi);

Este utilizatã pentru afisarea nodurilor heapu-lui.

Functia heapwalk receptioneaza un pointer catre o structura de tip heapinfo:

```
struct heapinfo {
void *ptr;
unsigned int size;
int in_use;
};
```

unde : ptr - pointer intors catre heapwalk (indica nodul in heap)

hi.size - reprezinta lungimea blocului in octeti

hi.in_use - este un flag careeste setat daca blocul este utilizat

Inainte de primul apel al functiei heapwalk trebuie ca hi.ptr=NULL.

Valoare intoarsa

Operatie cu succes : _HEAPEMPTY (= 1) heap vid;
_HEAPOK (= 2) heap corect (OK)
_HEAPEND (= 5) sfirsit heap

Operatie cu eroare : valoare <0

coreleft **<ALLOC.H>**

Declaratie: unsigned coreleft(void);

Functia coreleft intoarce dimensiunea memoriei RAM neutilizata.

3. Programe rezidente

Aceste programe se incarcă apoi se termina, dar rămîn în memoria calculatorului (se mai numesc programe *TSR - Terminate and Stay Resident*). Programele TSR sînt activate de o tastă prestabilită; de obicei aceste programe utilizează intreruperea asociată citirii tastaturii sau alte intreruperi ale sistemului. Aceste programe modifică tabela vectorilor de intrerupri astfel încît . la apariţia unei interuperi să poată fi eventual activate (se apelează codul rămas rezident in memorie).

Programele rezidente se încheie prin apelul unei funcţii ce apelează o funcţie DOS specială (terminate and stay resident).

Pentru limbajul C aceasta este funcţia :

void **keep** (*unsigned char* cod_iesire, *unsigned* dimensiune_rezervata) , cu antetul în DOS.H.

Eliminarea programelor TSR trebuie sã se efectueze în douã etape :

- restabilirea tabelei vectorilor de întrerupere
- eliberarea zonelor de memorie alocate programului TRS

Un exemplu de program utilitar care eliminã programe TSR (fãrã reincarcarea sistemului de operare) este RELEASE.EXE . Acest program foloseste informaţii de la programul MARK.COM.

Programul MARK.COM marcheazã zona de memorie unde se va incarca programul TSR şi memoreazã starea sistemului de intreruperi (în general starea calculatorului - ce poate fi modificatã de programul TSR).

Înainte de a se instala un program rezident se va da comanda MARK (acesta este el însuşi un program rezident).

Comanda RELEASE eliminã programul rezident pînã la MARK inclusiv.

Exemplu : Se instaleazã douã programe rezidente P1 şi P2 astfel :

MARK
P1
MARK
P2

Prima comandã RELEASE eliminã programul P2 , iar urmãtoarea comandã RELEASE eliminã programul P2.

4. Exemple de programe pentru managementul memoriei

```
/***********************************/
// aloc1.c - aloca memorie in heap
/***********************************/

#include <stdio.h>
#include <string.h>
#include <alloc.h>
#include <process.h>

void main(void)
{
   int n;
   char *s;

   n=10*1024;

   clrscr();
   printf("Heap:\t%u\n",coreleft());

   //aloca memorie in heap

   if ((s = (char *) malloc(n)) == NULL)
```

```
    {
        printf("Memorie insuficienta\n");
        exit(1);  /* iesire din program */
    }
  printf("Memoria a fost alocata\n");
  printf("Heap:\t%u\n",coreleft());
  system("mem /m alocl"); // apel comanda DOS
  system("mem /f");      // apel comanda DOS
  getch();
  free(s);
  printf("Memory was released\n");
  printf("Heap:\t%u\n",coreleft());
  system("mem /m alocl");
  system("mem /f");
  getch();

}

/****************************************/
//heapw.c - afiseaza nodurile heap-ului
/****************************************/

#include <stdio.h>
#include <conio.h>
#include <alloc.h>
#include <dos.h>

#define NUM_PTRS  10
#define NUM_BYTES 16
int main( void )
```

```
{
   struct heapinfo hi;
   char *array[ NUM_PTRS ];
   int i;

   for( i = 0; i < NUM_PTRS; i++ )
      array[ i ] = (char *) malloc( NUM_BYTES );

   for( i = 0; i < NUM_PTRS; i += 2 )
      free( array[ i ] );

   clrscr();

   hi.ptr = NULL;
   printf( "Adresa nod   Marime   Stare \n" );
   printf( "             ------   ------\n" );
   while( heapwalk( &hi ) == _HEAPOK )
   {
   printf(
"%x\:%x",FP_SEG(hi.ptr),FP_OFF(hi.ptr));
// FP_SEG - intoarce segmentul argumentului
// FP_off - intoarce offsetul argumentului
   printf( "%8u    %s\n", hi.size, hi.in_use ?
"used" : "free" );
   getch();
   }
   return 0;
}

/****************************************/
```

```
//aloc2.c - aloca blocuri de memorie DOS
/**************************************/

#include <dos.h>
#include <alloc.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <process.h>

#define n 53

char str1[]="ALOCAREA SEGMENTULUI 1 ***
ABCDEFGHIJKLMNOPQRSTUVXZY";
char str2[]="ALOCAREA SEGMENTULUI 2 ***
abcdefghijklmnopqrstuvxzy";

char far *p;
char far *p1;

int main(void)
{
  unsigned int size, segp , segp1;
  int i,stat;

  size = 64;           /* (64 x 16) = 1024 bytes */

  stat = allocmem(size, &segp);

  if (stat == -1)
```

```
    printf("Alocare - OK\n");
  else
    printf("Alocare - Fail\n");

  p=MK_FP(segp,0); // MK_FP - creaza un pointer
segp:0
  printf("Adresa segmentului alocat :
%x\n",FP_SEG(p));
  printf("Offsetul segmentului alocat :
%x\n",FP_OFF(p));

  for (i=0;i<n;i++,p++) {*p=str1[i];}

  system("mem /m aloc2");
  system("debug");
/* comenzi debug :    dxxxx:yy - afisare
                 exxxx:yy - afisare cu modificare
(SPACE pentru avans,              ENTER pentru
iesire
                      q      - exit        */

  stat = allocmem(size * 4,&segp1);

  if (stat == -1)
    printf("Alocare - OK\n");
  else
    printf("Alocare - Fail\n");

  p1=MK_FP(segp1,0);
```

```
  printf("Adresa segmentului alocat :
%x\n",FP_SEG(p1));
  printf("Offsetul segmentului alocat :
%x\n",FP_OFF(p1));

  for (i=0;i<n;i++,p1++) {*p1=str2[i];}

  system("mem /m aloc2");
  system("debug");

  freemem(segp);
  freemem(segp1);

  return 0;
}

/*******************************************************
******/
//tsr1.c - instaleaza / dezinstaleaza programe
rezidente
/*******************************************************
******/

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <process.h>

char key;
int cnt;
```

```
void main(void)
{

   while (key!='q')
   {
   system("mem /m vsafe");
   system("mem /m msd");
   cnt=cnt++;
   cnt=cnt%10;
   printf("%d\tTSR1***\n",cnt);;
   printf("(q - Quit, a - activate VSAFE, d -
deactivate VSAFE)");
   printf("(A - activate MSD)\n");
   key=getch();
   if (key=='a') system("vsafe");
   if (key=='d') system("vsafe/u");
   if (key=='A') system("msd");

   }
}


/*********************************************/
// tsr2.c - aloca memorie si ramine rezident
/*********************************************/


#include <stdio.h>
#include <conio.h>
#include <dos.h>
```

```
#include <stdlib.h>

char key;
int n,safety_space;

void main(void)
{

   while (key!='t')
   {
   printf("TSR2---\t k:0-9 - aloca k*16Ko de
memorie\n");
   printf("TSR2--- t - quit\n");
   key=getch();
   if ((key>=0x30)&&(key<=0x39)) n=key-0x30;
   safety_space=n*16*1024;
   if (key=='t') {keep(0, (_SS +
((_SP+safety_space)/16) - _psp));}
   }
}

/*********************************/
// tsr21.c - instaleaza pe tsr2
/*********************************/

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
```

```
char key;

void main(void)
{
   while (key!='q')
   {
   printf("a- activeaza TSR2---\n");
   key=getch();
   if(key=='a') {system("tsr2");}
   system("mem /m tsr2");
   }
}

/***********************************************/
// tsr.c - program rezident ce utilizeaza
intreruperi
/***********************************************/

#include <dos.h>
#define ATTR 0x0F00 //atribut video (alb pe negru)
/* ATTR = BFFFCCCC00000000 ;
              B=afisare continua(0)/intermitenta(1)
              FFF - culoare fundal
              CCCC - culoare text
*/

/* Intreruperea de la ceasul de timp real - 50ms
*/
#define INTR 0x1C
```

```
/* se reduce marimea stivei si a heap-ului pentru
a crea un program redus*/
extern unsigned _heaplen = 1024;
extern unsigned _stklen  = 512;

//rutina de servire a intreruperii veche
void interrupt (*oldhandler)(void);

typedef unsigned int (far *s_arrayptr);

// noua rutina de servire a intreruperii
void interrupt handler()
{
   s_arrayptr screen[25]; //vector de pointeri
catre linii
   int count,cnt;

 /* Creaza un pointer catre prima locatie a
memoriei video: B800:0000 */
   screen[0] = (s_arrayptr) MK_FP(0xB800,0);

 /* incrementeaza count modulo 10 */
   count++;count %= 10;
 /*afiseaza count */
   screen[0][79] = count + '0'+ ATTR;
 //[linie][coloana]

 /* apel vechea rutina de intrerupere */
 oldhandler();
 }
```

```
void main(void)
{

/* preia adresa de servire a intreruperii vechi*/
oldhandler = getvect(INTR);

/* instaleaza noul vector de intrerupere */
setvect(INTR, handler);

/* _psp este adresa de start a programului in
memorie
   Virful stivei este sfirsitul programului
   Utilizind _SS si _SP impreuna se poate
determina sfirsitul
   stivei (plus un spatiu de rezerva) :
     (_SS + ((_SP + safety space)/16) - _psp)
*/
//termina programul si ramine rezident
keep(0, (_SS + (_SP/16) - _psp));

}


/**************************************************/
// remove.c - elimina programe rezidente (in
anumite conditii)
/**************************************************/

#include <stdio.h>
```

```
#include <conio.h>
#include <dos.h>
#include <process.h>
#include <string.h>

unsigned int seg;
char nume[20];
char command[20];

void free_block(unsigned int s)
{
unsigned char far *p;

p=MK_FP(s,0);                    //pointer catre
inceputul blocului

  printf("Adresa segmentului alocat :
%x\n",FP_SEG(p));
  printf("Offsetul segmentului alocat :
%x\n",FP_OFF(p));

p++;                               // pointer catre al
doilea element
*p=0;                              // stergere
p++;                               // pointer catre al
treilea element
*p=0;                              // stergere
}

void main(void)
```

```
{
   clrscr();
   printf("ATENTIE !!!! ADRESELE DE SEGMENT ALE
BLOCURILOR ALOCATE \n");
   printf("PROGRAMULUI TREBUIE INTRODUSE CORECT
!!!!\n");
   printf("\nATENTIE !!!! NU INTRODUCETI ALTE
ADREȘELE DE SEGMENT \n");
   printf("IN AFARA CELOR SPECIFICATE  !!!!\n");
printf("\n");
   printf("ELIMINAREA PROGRAMULUI REZIDENT ESTE
CORECTA NUMAI DACA \n");
   printf("ACESTA NU UTILIZEAZA INTRERUPERI \n");
   printf("\nIesirea din program cu adresa de
segment : FFFF \n");

   printf("Numele programului :");
   scanf("%s",&nume);
   strcat(command,"mem /m ");
   strcat(command,nume);
   while (1)
   {
   system(command);
   printf("Adresa de segment a blocului :");
   scanf("%x",&seg);
   printf("\n");
   if(seg!=0xffff) free_block(seg); else exit(0);
   }
 }
```

5. Desfăsurarea lucrării

a) Să se studieze tipurile de memorie într-un calculator PC.

b) Să se verifice functionarea comenzilor DOS: MEM.EXE (comanda MEM /?) şi MSD.EXE
(utilitar ce poate afisa alocarea memoriei convenţionale)

c) Să se studieze exemplele de programe pentru alocarea memoriei

d) Să se observe , cu ajutorul programului MEM , diferenţa dintre alocarea în heap şi alocarea de blocuri de memorie DOS.

e) Să se verifice functionarea programelor MARK.EXE şi RELEASE.EXE

Temă

Să se realizeze un program care alocă 3 blocuri de memorie DOS , fiecare de cîte 1ko, apoi eliberează unul din aceste blocuri , crează un bloc de memorie DOS de 2ko, copiază informaţia din cele două blocuri de memorie rămase în acest nou bloc , apoi eliberează blocurile de 1ko. (Programul realizează o compactare a spaţiului de memorie). Se vor utiliza, ca exemplu , programele ALOC2.C şi REMOVE.C .

Sisteme multiprocesor. Conectarea si sincronizarea procesoarelor sistemului multiprocesor

Scopul lucrării

a) Principii de interconectare a procesoerelor în sisteme multiprocesor.
b) Descrierea unui sistem multiprocesor cu două procesoare interconectate prin memorie comună.
c) Sincronizarea software a activitãtilor procesosrelor sistemului multiprocesor.

1. Arhitecturi de sisteme multiprocesor

Existã două categorii de sisteme multiprocesor : arhitecturã *centralizatã* si arhitecturã *distribuitã* .Cele două solutii constructive sînt ilustrate în figura 1.

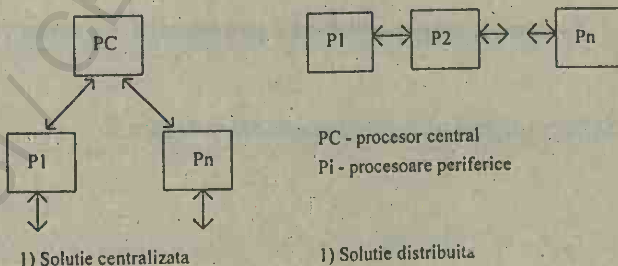


Figura 1. Arhitecturi de sisteme multiprocesor

Solutia centralizată este utilizată pentru sisteme complexe. Este necesar un mecanism de intercomunicare între procesoare (realizat software sau hardware) care limitează performantele sistemului.

Pentru solutia distribuită deciziile se iau local de către procesoarele periferice. Mecanismul de intercomunicare (uzual realizat software) este mai simplu. Este necesară divizarea functiilor sistemului în subfunctii bine determinate care sînt atribuite procesoarelor locale.

În practică se utilizează si solutii mixte , cu un procesor central si mai multe procesoare locale.

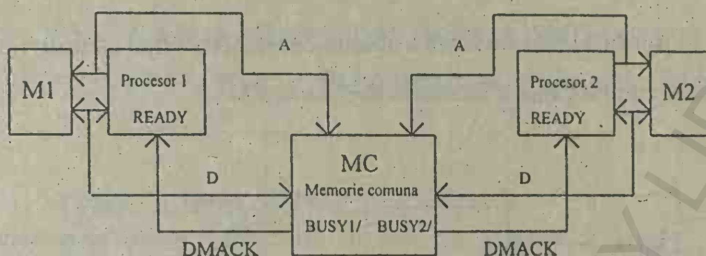
1.1. Solutii de interconectare

Principalele moduri de conectare între două procesoare sînt :

- cu memorie comună
- cu împărtirea busurilor

1.1.1. Sistem multiprocesor cu 2 procesoare cu memorie comună

Schema sistemului este prezentată în figura 2.



Semnalele de adresa si control pentru memoria comuna nu s-au figurat
MC - (IDT7132 - 2k x 8 dual cu circuitul de arbitrare inclus)
M1, M2 - memorii privata A , D - adrese , date
DMACK - Data Acknowledge (trebuie sa fie 1 pentru functionare normal

Figura 2. Sistem multiprocesor cu memorie comunã

Memoria comunã este un circuit mai complex ce include memoria propriu-zisã si un controler care arbitreazã conflictele de acces ce pot apare dacã cele douã procesoare cer accesul "simultan" la memoria comunã.

Conflictele de acces pot apare dacã un procesor citeste iar celãlalt scrie aceeasi locatie de memorie sau dacã ambele procesoare scriu în aceasi locatie de memorie.

Circuitul de arbitrare a conflictelor de acces are urmãtoarele functii :

- comparã adresele generate de cele douã procesoare pentru memoria comunã
- genereazã semnalele de control pentru memoria comunã (de exemplu WE/)
- genereazã semnale pentru blocarea unui ciclu de acces (cel sosit putin mai tîrziu); se genereazã de exemplu un semnal BUSY/ .

În figura 3. este ilustrată o solutie constructivă fără controler integrat (memoria comună este o memorie RAM uzuală).

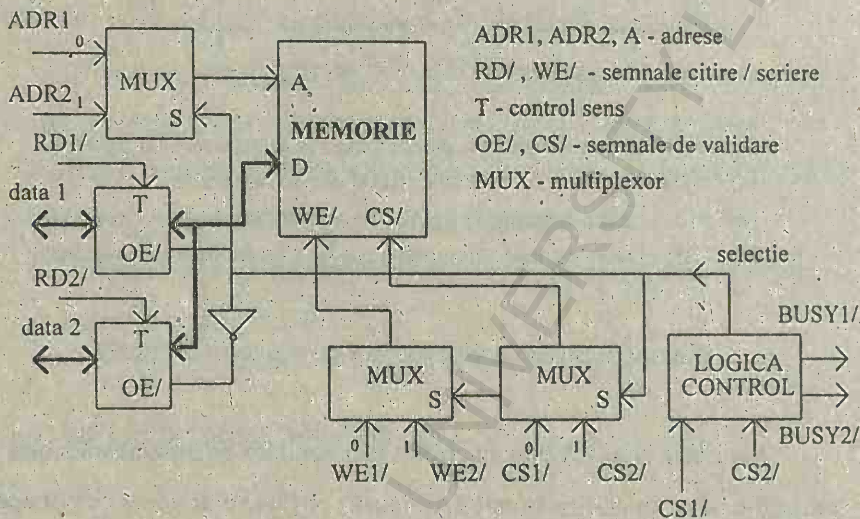


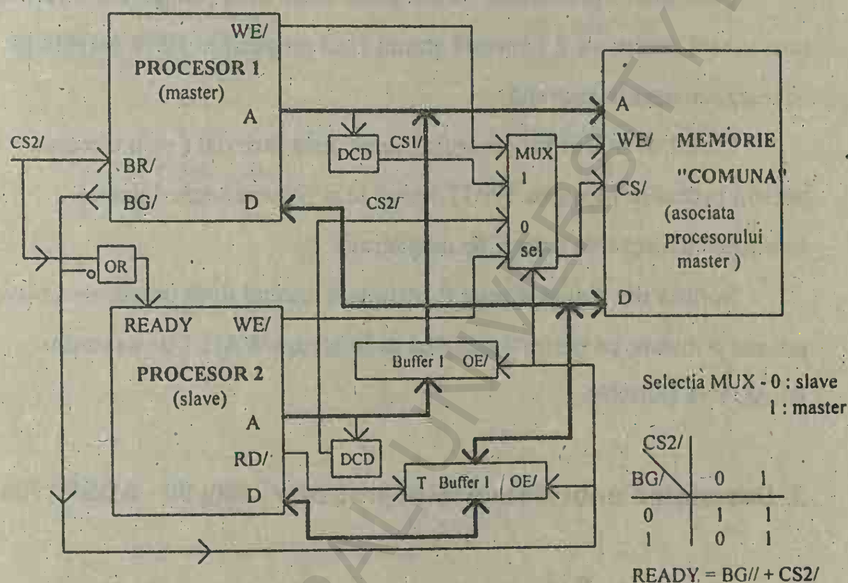
Figura 3. Interconectarea procesoarelor prin memorie comună (fãra controler integrat)

Dezavantajul solutiei din figura 3 este complexitatea hardware în raport cu solutia integrată din figura 2.

1.1.2. Sistem multiprocesor cu împãrtirea bus-urilor (*bus shareing*)

În figura 4.. se prezintă o solutie de interconectare ce utilizează principiul cererii - achitării de magistrală (*bus request - bus grant*).

În caz contrar se pot insera stãri de WAIT pentru ciclurile de acces la memoria comunã pentru a astepta eliberarea magistralelor de cãtre *master* (ca în figura 5.) .



a) schema bloc de principiu



b) diagrama de timp

Figura 4. Sistem multiprocesor cu împãrtirea busurilor

Comunicarea între procesoare se realizează tot printr-o memorie comună; cele 2 procesoare utilizează aceleasi bus-uri pentru accesul la memoria comună.

Unul dintre procesoare va fi *master* pînă cînd primeste o cerere de bus; acestă cerere va fi generată atunci cînd procesorul *slave* doreste să acceseze memoria comună.

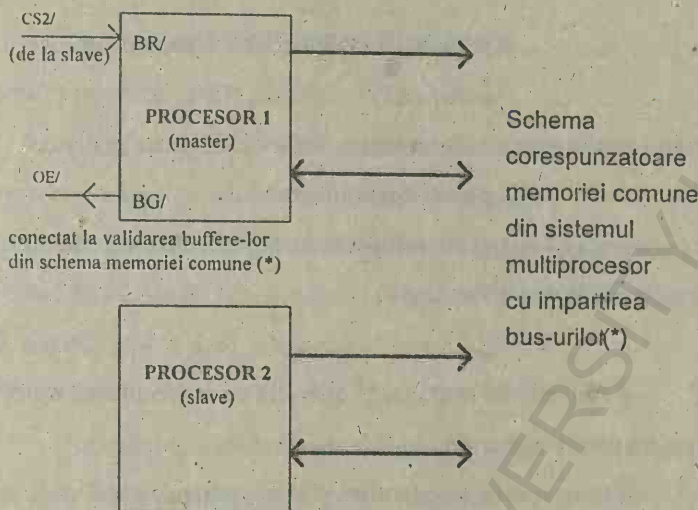
Ciclul de bus al procesorului *slave* este întîrziat (prin trecerea acestui procesor în starea WAIT) pînă la activarea semnalului ce semnifică acceptarea cererii de magistrală.

Solutia din figura 4. este functională numai dacă procesorul *slave* posedă o intrare ce permite trecerea sa în starea WAIT (de exemplu READY la I80x86).

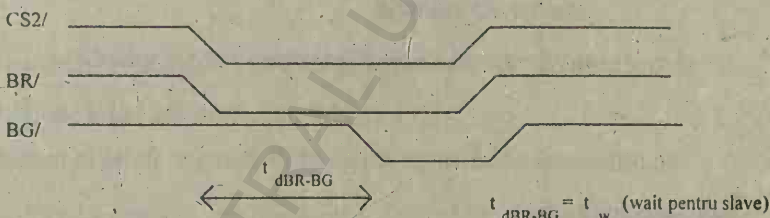
2. Descrierea unui sistem multiprocesor - I80x86 - ADSP210x

Pentru exemplificarea sistemului multiprocesor cu împărtirea busurilor se consideră un sistem cu procesor specializat (procesor DSP - Digital Signal Processing - ADSP210x) interconectat cu procesorul I80x86 din structura unui calculator PC.

Conectarea între aceste două subsisteme se realizează prin memorie comună (situată pe placa cu procesor DSP). Memoria comună este memoria procesorului DSP (considerat master) , dar poate fi accesată de către PC prin mecanismul *bus reguest - bus grant* , descris în figurile 4 sau 5.



a) schema bloc de principiu



b) diagrama de timp

Figura 5. Varianta a unui sistem multiprocesor cu împãrtirea busurilor

Zona de memorie comunã (de interes în acest caz) este "vãzutã" în felul urmãtor de cãtre procesorul DSP sau de cãtre procesorul I80x86 (PC):

- pentru DSP : memoria comunã este pe 16 biti între adresele 0-1FFF
- pentru PC : memoria comunã pe 8 biti atfel :

D000:6000 - D000:7FFF - octetul cel mai putin semnificativ

D000:8000 - D000:9FFF - octetul cel mai semnificativ

În mod suplimentar sistemul cu procesor DSP poate genera o întrerupere către PC (utilă pentru sincronizare).

Se consideră că sistemul multiprocesor astfel constituit trebuie să îndeplinească următoarele functiuni :

PC - stabileste doi operanzi si operatia ce se efectuează asupra acestora (în memoria comună)

- citeste rezultatul operatiei (din memoria comună) si îl afisează pe display

DSP - citeste (din memoria comună operanzii si tipul operatiei

- efectuează operatia

- stochează (tot în memoria comună) rezultatul

Se utilizează următoarea structură de date (de finită în memoria comună) (figura 6):

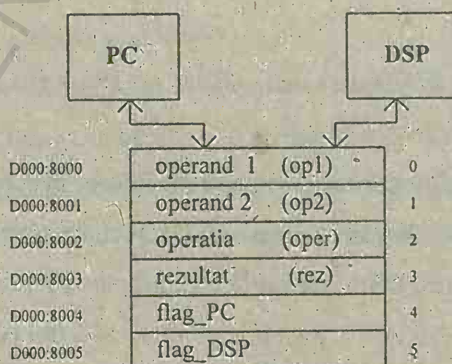


Figura 6. Structura de date asociată sistemului multiprocesor

Se consideră operanzii pe 8 biti (octetii cei mai putin semnificativi pentru operanzii DSP sînt egali cu zero).

Ultimele două cîmpuri ale structurii de date vor fi utilizate pentru sincronizarea activitătilor celor două procesoare.

Cererea de bus (BR/) se efectuează prin scrierea unui 0 logic în portul de adresă 0x301.Anularea cererii de bus se efectuează prin scrierea în acelasi port a unui 1 logic.

Se consideră că cerearea de bus este acceptată imediat (diferentele de viteză de executie între DSP si PC sînt mari , semnalul BG/ - acceptarea cererii de bus - se va activa înainte ca I80x86 să efectueze ciclul de bus cerut) . (Conectarea este realizată după ideea din figura 5.)

3. Sincronizarea activitătilor procesoarelor din sistemul multiprocesor I80x86 - ADSP210x

Organigramele de functionare pentru cele două procesoare din sistemul multiprocesor sînt ilustrate în figura 7.

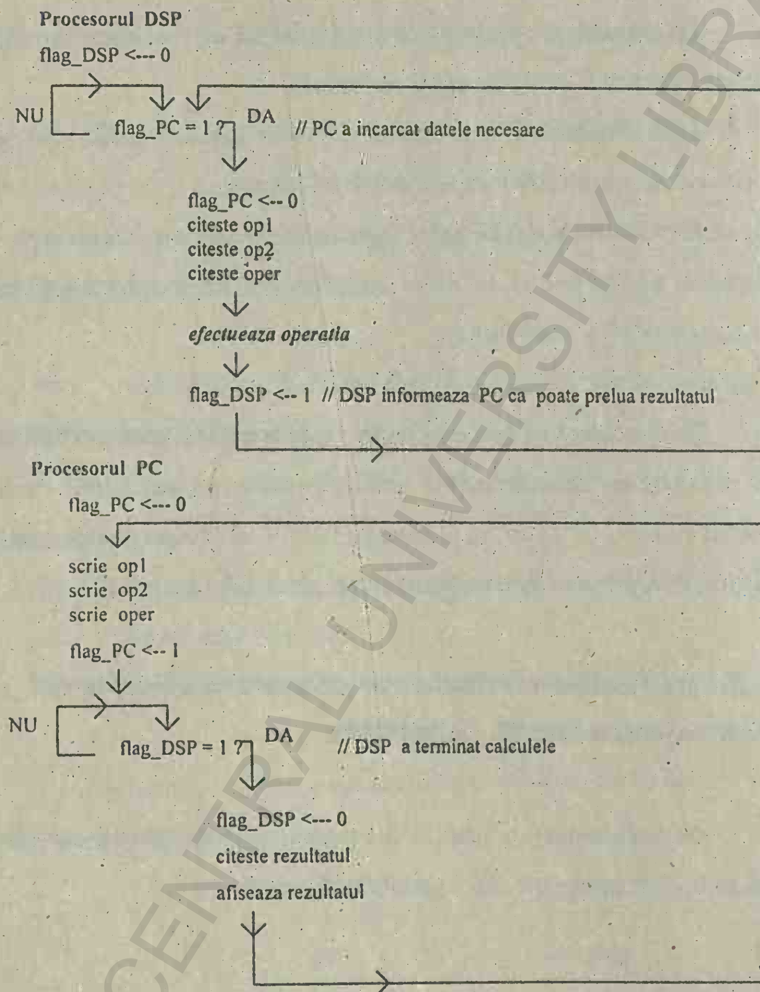

Procesorul DSP
flag_DSP <--- 0
NU
flag_PC = 1 ?
DA
// PC a incarcat datele necesare
flag_PC <-- 0
citeste op1
citeste op2
citeste oper
efectueaza operatia
flag_DSP <-- 1 // DSP informeaza PC ca poate prelua rezultatul
Procesorul PC
flag_PC <--- 0
scrie op1
scrie op2
scrie oper
flag_PC <-- 1
NU
flag_DSP = 1 ?
DA
// DSP a terminat calculele
flag_DSP <--- 0
citeste rezultatul
afiseaza rezultatul

Figura 7.

4. Programe aplicative pentru sistemul multiprocesor

```
// ASC3.C - program pentru I80x86 - PC

#include <conio.h>
#include <stdio.h>
#include <dos.h>
#include <stdlib.h>

char cmd[6]={0,0,0,-1,0,0};
/*
cmd[0] <--- operand 1
cmd[1] <--- operand 1
cmd[2] <--- operatia astfel :
             0 - operand1 + operand 2 = rezultat
             1 - operand1 - operand 2 = rezultat
             2 - operand1 * operand 2 = rezultat

cmd[3] <--- rezultat
cmd[4] <--- flag_PC
cmd[5] <--- flag_DSP
*/

char unsigned far *p1;
char unsigned far *p2;

void wbuff(char c,char p)
// scrie caracterul c in buffer-ul comun
// la adresa data de indexul p
```

```
{
outport(0x301,0x0100); // BR/ activat
p1=(char unsigned far*)MK_FP(0xD000,0x6000+p);
 // octetul LSB
p2=(char unsigned far*)MK_FP(0xD000,0x8000+p);
// octetul MSB
 *p1=c;
 *p2=0;  // octetul MSB este 0
outport(0x301,0x0101); // BR/ dezactivat
}

char rbuff(char p)
// intoarce caracterul citit din buffer-ul comun
// la adresa data de indexul p
{
char c1,c2;
outport(0x301,0x0100); // BR/ activat

p1=(char unsigned far*)MK_FP(0xD000,0x6000+p); //
octetul LSB
p2=(char unsigned far*)MK_FP(0xD000,0x8000+p); //
octetul MSB

  c1=*p1;
  c2=*p2;

 outport(0x301,0x0101); // BR/ dezactivat

 return(256*c2 + c1);
 }
```

```
void main(void)
{
char key;
clrscr();
while(key!='x')
{
// scriere operanzi
printf("Operandul 1 =  " );
scanf("%d",&cmd[0]);
wbuff(cmd[0],0);
printf("Operandul 2 =  " );
scanf("%d",&cmd[1]);
wbuff(cmd[1],1);
// scriere operatie
printf("Operatia (0 - adunare, 1 - scadere, 2 -
inmultire) =  " );
scanf("%d",&cmd[2]);
wbuff(cmd[2],2);
// flag_PC=1
cmd[4]=1;
wbuff(1,4);      // scriere  in bufferul comun

//===== Sectiunea A ======
// intirziere pentru ca DSP sa poata efectua
prelucrarea
//fara a fi oprit de o cerere de bus
//delay(1);
//==== Sectiunea A ======
```

```
//===== Sectiunea B ======
// test flag_DSP=1 si flag_PC=0
// DSP a preluat datele si a efectuat operatia
///*
while((cmd[5]==0)||(cmd[4]==1))
{
cmd[5]=rbuff(5);
cmd[4]=rbuff(4);
}
//*/
//===== Sectiunea B ======
// pozitioneaza flag_DSP=0

cmd[5]=0;
wbuff(0,5);      // scriere  in bufferul comun
// citeste rezultatul
cmd[3]=rbuff(3);
printf("\nRezultatul este =  %d\n\n",cmd[3]);
printf("x - Exit\tEnter - Continue\n\n");
key=getch();
}
}
// RK.C - program pentru procesorul DSP


/*
cmd[0] <-- operand 1
cmd[1] <-- operand 2
cmd[2] <-- operatia
cmd[3] <-- rezultatul
```

```
cmd[4] <-- flag_PC
cmd[5] <-- flag_DSP
*/
/* functia de adunare */
int add(int a,int b)
{
return(a+b);
}
/* functia de scadere */
int sub(int a,int b)
{
return(a-b);
}
/* functia de inmultire */
int mpy(int a,int b)
{
return(a*b);
}
int op1; /* operand 1 */
int op2; /* operand 2 */
int oper; /* operatia */
int rez; /* rezultatul */
int flag_PC; /* flag pentru PC */
int flag_DSP; /*flag pentru DSP */

void main(void)
{
asm(".var/dm/abs=0   cmd[6];");
asm(".init cmd[0] : 0,0,0,-1,0,0;");
```

```
while (1)
{
asm("    ax0=dm(cmd+4);\
     dm(flag_PC_)=ax0;"); /* flag_PC <-- cmd[4] */

if (flag_PC==1)
{
flag_PC=0;
asm("    ax0=dm(flag_PC_);\
     dm(cmd+4)=ax0;"); /* cmd[4] <-- flag_PC */
     asm("    ax0=dm(cmd);\
     dm(op1_)=ax0;"); /* op1 <-- cmd[0] */
     asm("    ax0=dm(cmd+1);\
     dm(op2_)=ax0;"); /* op2 <-- cmd[1] */
     asm("    ax0=dm(cmd+2);\
     dm(oper_)=ax0;"); /* oper <-- cmd[2] */

switch (oper)
{
case 0: {rez=add(op1,op2);break;}
case 1: {rez=sub(op1,op2);break;}
case 2: {rez=mpy(op1,op2);break;}
default : rez=0;
}
asm("    ax0=dm(rez_);\
     dm(cmd+3)=ax0;"); /* cmd[3] <-- rez */
flag_DSP=1;
asm("    ax0=dm(flag_DSP_);\
     dm(cmd+5)=ax0;"); /* cmd[5] <-- flag_DSP */
}
```

```
}
}
```

5. Desfãsurarea lucrãrii

a) Sã se studieze programele ASC3.C si RK.C (dupã organigrama din figura 7).

b) Se va încãrca programul RK.EXE în memoria procesorului DSP cu comenzile (de la prompter-ul DOS) :

LOAD RK.EXE

apoi se vor specifica parametrii :

Segment address 0,1,2,3,4 : 3

I/O address 0,1,2,3 : 0

Dupã încãrcare procesorul DSP executã automat programul.

Se lanseazã , din mediul integrat Borland C , programul ASC3.C .

Se va urmãri executia programelor.

c) Sã se execute programul ASC3.C cu sectiunea A sau cu sectiunea B (cu întîrziere sau cu sincronizare prin falg-uri). Explicati functionarea în cele douã cazuri.

Temã

Sã se modifice organigrama din figura 7 (pentru PC) astfel încît sincronizarea sã se efectueze utilizînd întreruperi de la DSP (generate cînd DSP a terminat prelucrãrile).

Organizarea memoriei secundare (virtuale) în PC. Funcţii de management al directoarelor şi fişierelor

Scopul lucrării

a) Studiul principalelor tipuri de memorie secundară.

b) Managementul directoarelor şi fişierelor utilizând funcţii de nivel înalt.

4.1. Tipuri de memorie secundară

Memoria principală a unui sistem de calcul trebuie să fie accesibilă direct şi cât mai rapid posibil. Totuşi, preţul ridicat al acesteia face necesară utilizarea unei memorii de mai mare capacitate (*de masă*), care să fie mai ieftină. Această memorie, numită *secundară* sau *virtuală* are dezavantajul unei viteze mai mici de acces la informaţii decât memoria principală.

Benzile magnetice

Istoric vorbind, benzile magnetice sunt primele memorii secundare ale sistemelor de calcul. Benzile magnetice informatice sunt asemănătoare celor utilizate de către magnetofoane. Principiul de funţionare al unităţilor de bandă magnetică este deplasarea unei benzi cu viteză constantă în faţa unui cap de citire/scriere. Pentru scrierea unei informaţii pe bandă este de

ajuns să se varieze curentul electric în capul magnetic, ceea ce provoacă o magnetizare locală a părţii de bandă care se găseşte sub cap.

Figura 1 arată organizarea tipică a informaţiei pe o bandă magnetică. Cel mai adesea informaţiile sunt înregistrate sub forma unor cuvinte de un caracter succesive, pe un anumit număr de piste adiacente, fiecare caracter cuprinzând în general 8 biţi plus un bit de paritate, pentru a ameliora fiabilitatea înregistrării. Densitatea tipică a înregistrării este de 1600 bpi (*byte per inch*). Aceasta corespunde înregistrării unui cuvânt pe cca. 0,03 mm. Alte densităţi utilizate sunt 800 bpi şi 6250 bpi.

După terminarea scrierii unei înregistrări fizice, adică a unui bloc de date, se lasă un *spaţiu liber de înregistrare* (un *gap*) pe bandă înainte de a se scrie un alt bloc. Acest spaţiu permite benzii să atingă şi să anuleze viteza nominală a citirii/scrierii pe durata pornirii şi respectiv a opririi defilării benzii prin faţa capului de citire. Dacă se scriu date de mică dimensiune pe bandă, se obţine un număr mare de *gap*-uri, banda fiind utilizată ineficient.

Benzile magnetice sunt memorii secundare cu *acces secvenţial.* Atunci când banda este poziţionată la început, citirea blocului de date *n* necesită citirea secvenţială prealabilă a blocurilor de la 1 la n-1. Dacă informaţia dorită este aproape de finalul benzii, programul trebuie să citească aproape toată banda înainte de obţinerea informaţiei, ceea ce poate lua câteva minute. A face un procesor care poate executa milioane de instrucţiuni pe secundă să aştepte câteva sute de secunde este o aberaţie. De aceea banda magnetică e utilizată pentru aplicaţii secvenţiale sau pentru arhivare.

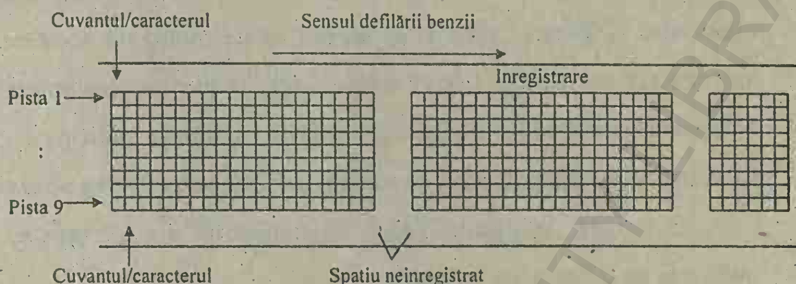


Fig. 1. Organizarea înregistrării pe o bandă magnetică

Discurile magnetice

Un disc magnetic este un platou metalic circular din material non magnetic (ex. cupru, aluminiu) cu diametru de la 15 la 30 cm, care este acoperit pe cele două feţe cu o substanţă magnetizabilă (figura 2).

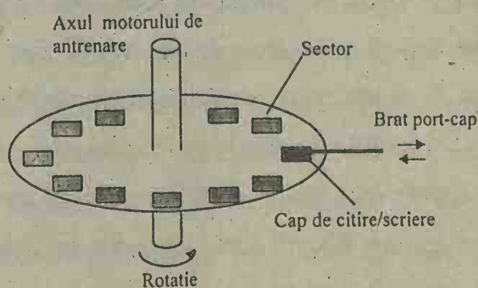


Fig. 2. Disc cu un platou

Informaţiile sunt înregistrate pe un anumit număr de *piste* concentrice. Discurile au câteva zeci sau sute de piste pe faţă. În general, în unităţile de discuri, capetele de citire/scriere sunt plasate pe un braţ mobil

care se deplasează radial pe disc. Adesea, unităţile de discuri cuprind mai multe discuri dispuse vertical pe acelaşi ax de rotaţie. În acest caz, braţul are câte un cap pentru fiecare dintre feţele discurilor, deplasarea braţului antrenând simultan ansamblul de capete. Distanţa radială a capetelor (distanţa de la capete la axul de rotaţie) defineşte un *cilindru de date*.

O unitate de discuri dotată cu n platouri are $2n$ capete, ceea ce defineşte un cilindru cu $2n$ piste.

Pistele sunt divizate în sectoare, de ordinul zecilor pe o pistă. Un sector constituie unitatea de informaţie pe un disc, şi are în general capacitatea de 512 octeţi. Pentru definirea unui schimb de informaţii cu discul, un program trebuie să precizeze: *cilindrul* şi *capul* care definesc *pista*, *numărul de sector* la care începe înregistrarea, *numărul de cuvinte* schimbate, *adresa de memorie* care primeşte sau furnizează informaţia.

Transferul între disc şi memorie debutează întotdeauna cu începutul unui sector. Când se transferă informaţii voluminoase care necesită mai multe sectoare de pe piste adiacente ale aceluiaşi cilindru nu se pierde timp cu trecerea de la un cap de citire la altul. Dacă însă transferul necesită trecerea de la un cilindru la altul, este necesară deplasarea braţului port-capete, ceea ce conduce la pierdere de timp (de ordinul ms sau zecilor de ms). Pe de altă parte, viteza de rotaţie tipică este de 3600 rotaţii/minut.

Unităţile de disc ale calculatoarelor personale (PC) cuprind frecvent un număr de platouri plasate pe acelaşI ax de rotaţie (figura 3). Acestea sunt numite discuri dure (*hard-disk*-uri), datorită materialului metalic al suportului utilizat de platouri.

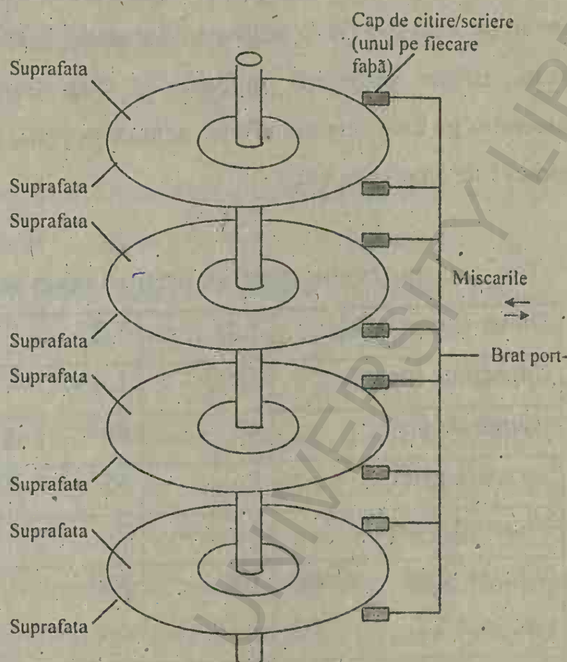


Fig. 3. Disc cu patru platouri

Pentru transportul programelor între PC-uri au fost introduse discuri mici, numite dischete sau discuri suple (*floppy-disk*-uri), datorită materialului plastic al suportului. Pentru a reduce uzura dischetelor, capetele de citire/scriere se retrag şi rotaţia discului este întreruptă pe durata inactivităţii, ceea ce duce la întârzieri în accesul la informaţii. Dimensiunile actuale ale dischetelor sunt de 5,25" (din ce în ce mai rar utilizată) şi 3,5". Fiecare tip are versiuni cu densitate de înregistrare normală şi mărită. Parametrii acestora sunt prezentaţi în tabelul 1.

Pentru initializarea discurilor este necesara mai intai o *formatare la nivel fizic*, de nivel jos, care pregateste discul pentru formatarea propriu-zisa.

Formatarea de nivel inalt imparte discul in piste si sectoare. Sistemul de operare DOS utilizeaza comanda FORMAT pentru aceasta operatie. Exista programe utilitare de management al fisierelor si directoarelor pe disc care pot efectua aceasta operatie (Norton Commander, Windows File Manager, etc.).

Tab. 1. Caracteristici specifice pentru 4 tipuri de dischete

Dimensiune (inch)	5,25	5,25	3,5	3,5
Capacitate (octeti)	360 k	1,2 M	720 k	1,44 M
Numar de piste	40	80	80	80
Sectoare/pista	9	15	9	18
Numar de capete	2	2	2	2
Viteza de rotatie (rot./min.)	300	360	300	300
Debitul (kb/s)	250	500	250	500
Tipul suportului	suplu	suplu	rigid	rigid

In cazul discurilor hard, formatarea de nivel inalt poate fi urmata de *partitionarea discului fizic* in mai multe *discuri virtuale*. Sistemul de operare DOS utilizeaza comanda FDISK pentru aceasta operatie.

4.2. Managementul directoarelor şi fişierelor

Spatiul pe disc al PC-urilor este alocat fisierelor in unitati de alocare numite *cluster*-e. Dimensiunea cluster-elor variaza in functie de tipul discului, formatul si dimensiunea lui. Floppy-disk-urile au in general dimensiunea clusterelor de 1k octeti, pe cand dimensiunea clusterelor hard-

disk-urilor poate varia intre 512bytes si 16kbytes, depinzand de capacitatea discului. Pe un hard-disk pot exista zeci de mii de cluster-e. Comanda FORMAT determina aceasta dimensiune.

Cluster-ele sunt numerotate si un fisier pe disc este format din unul sau mai multe astfel de unitati de alocare. In *tabela de alocare a fisierului* (FAT = *File Allocation Table*) se pastreaza numerele si ordinea clusterelor apartinand unui fisier, data si momentul crearii sale.

Fragmentarea este consecinta naturala a crearii si stergerii fisierelor. Deoarece un fisier este format dintr-unul sau mai multe cluster-e a caror ordine este cunoscuta, un fisier poate fi plasat oriunde pe disc. Un procesor de texte poate de exemplu sa resalveze un fisier modificat scriind in alta parte pe disc, eliberand apoi spatiul utilizat initial pentru alte fisiere.

Pentru a citi din fisier sau scrie in fisier, sistemul de operare DOS urmeaza pur si simplu lista curenta a numerelor de cluster-e din FAT. Nu conteaza cate parti sunt si nici de cat de dispersate sunt pe disc. Sarind pe toata suprafata discului pentru a citi sau scrie biti se reduce viteza de acces. Daca partile unui fisier nu sunt plasate alaturat, fisierul este numit *fragmentat*, iar operatiile de refacere in caz de distrugere, stergere sau pierdere sunt ingreunate.

Utilitarul TMAP prezinta nivelul fregmentarii tuturor fisierelor de pe disk.

4.3. Exemple de funcţii C pentru managementul directoarelor şi fişierelor

getfat : Obtinerea informatiilor FAT

Declaratie: void getfat(unsigned char drive, struct fatinfo *dtable);

Parametru	*Semnificatie*
drive	specifica drive-ul ale carui informatii sunt obtinute (0 = implicit, 1 = A, 2 = B, etc.).
dtable	adresa structurii fatinfo

```
struct fatinfo {
  char  fi_sclus;     /* sectoare pe cluster */
  char  fi_fatid;     /* identificatorul FAT */
  int   fi_nclus;     /* numarul de cluster */
  int   fi_bysec;     /* octeti pe sector */
};
```

_dos_getdiskfree : Obtinerea spatiului liber pe disc

Declaratie: unsigned _dos_getdiskfree(unsigned char drive,
struct diskfree_t *dtable);

Parametru	*Semnificatie*
drive	specifica drive-ul ale carui informatii sunt obtinute (0 = implicit, 1 = A, 2 = B, etc.).
dtable	adresa structurii dfree

```
struct dfree {
  unsigned df_avail;  /* clustere disponibile */
  unsigned df_total;  /* total clustere */
  unsigned df_bsec;   /* octeti pe sector */
  unsigned df_sclus;  /* sectoare pe cluster */
};
```

getdisk : Obtinerea numarului driveului curent

setdisk : Setarea numarului driveului curent

Declaratii: int getdisk(void);
int setdisk(int drive);

Parametru	*Semnificatie*
drive	specifica drive-ul setat (0 = implicit, 1 = A, 2 = B, etc.).

mkdir : creaza un director

rmdir : sterge un director de fisiere DOS

Declaratii: int mkdir(const char *path);

int rmdir(const char *path);

Parametru *Semnificatie*

path sir de caractere specificand calea

Obs: Calea pentru *rmdir* trebuie sa nu fie directorul curent sau directorul radacina, iar directorul trebuie sa fie vid.

Valoare returnata: mkdir returneaza 0 daca directorul a fost creat

rmdir returneaza 0 daca directorul a fost sters

mkdir si rmdir returneaza -1 si seteaza *errno* astfel:

EACCES = operatie nepermisa

ENOENT = cale sau fisier inexistent

getcurdir : Obtine directorul curent pentru un drive specificat

Declaratie: int getcurdir(int drive, char *directory);

Parametru *Semnificatie*

drive specifica drive-ul setat (0 = implicit, 1 = A, 2 = B, etc.).

directory adresa unei zone de memorie (de lungime MAXDIR) in care va fi plasat un nume de director terminat cu null.

Valoare returnata: in caz de suces, returneaza 0

in caz de suces, returneaza -1

chdir : Schimba directorul curent

Declaratie: int chdir(const char *path);

Parametru *Semnificatie*

path	sir de caractere specificand calea noua

Valoare returnata: in caz de suces, returneaza 0

in caz de suces, returneaza -1 si seteaza *errno* astfel:

ENOENT = cale sau nume de fisier inexistente

_dos_creat : Creeaza fisier sau suprascrie unul existent (utilizand functia DOS 0x3C)

Declaratie: unsigned _dos_creat(const char *path, int attrib, int *handlep);

Parametru *Semnificatie*

attrib permisiune de acces (atribut DOS), una din constantele:

FA_NORMAL	fisier normal
FA_RDONLY	fisier read-only
FA_HIDDEN	fisier ascuns
FA_SYSTEM	fisier sistem

handlep adresa identificatorului logic al fisierului (*handle*)

path calea si numele fisierului (calea implicita este directorul curent)

_dos_write : Scrie intr-un fisier (utilizand functia DOS 0x40)

Declaratie: unsigned _dos_write(int handle, void far *buf, unsigned len, unsigned *nwritten);

Parametru	*Semnificatie*
handle	identificatorului logic al fisierului
nwritten	adresa numarului de biti curent scrisi
buf	adresa buffer-ului din care functia scrie octetii
len	numarul de octeti pe care functia se asteapta ii scrie

_dos_close : Inchide un fisier asociat cu un identificator logic de fisier (utilizand functia DOS 0x3E)

Declaratie: unsigned _dos_close(int handle);

_dos_open : Deschide un fisier pentru citire sau scriere (utilizand functia DOS 0x3D)

Declaratie: #include <fcntl.h>
#include <share.h>
#include <dos.h>
unsigned _dos_open(const char *filename, unsigned oflags, int *handlep);

Parametru	*Semnificatie*
filename	adresa numelui fisierului deschis
oflags	modul de deschidere, constante simbolice definite in FCNTL.H si tipul deschiderii, constante simbolice definite in SHARE.H
handlep	adresa identificatorului logic al fisierului (*handle*)

_dos_read : Scrie intr-un fisier (utilizand functia DOS 0x3F)

Declaratie: unsigned _dos_read(int handle, void far *buf, unsigned len, unsigned *nread);

Parametru	*Semnificatie*
handle	identificatorului logic al fisierului
nread	adresa numarului de biti curent cititi
buf	adresa buffer-ului in care functia citeste octetii
len	numarul de octeti pe care functia se asteapta ii citeasca

4.4. Exemple de programe

```
{FAT1.C - Obtinerea spatiului total pe disc}

#include <stdio.h>
#include <conio.h>
#include <dos.h>

void main(void)
{
struct fatinfo diskinfo;
unsigned int u;        //unitatea de disc dorita
unsigned int nc;       //nr de unitati de alocare
(cluster)
unsigned int ns;       //nr de sectoare pe cluster
unsigned int no;       //nr de octeti pe sector
long unsigned int dim;          //dimensiune disc

clrscr();
printf("Obtinerea informatiilor FAT\n");
printf("Unitate de disc : (0-implicita,1-A:,2-B:3-
C:....) : ");
scanf("%d",&u);
getfat(u, &diskinfo);
printf("\n");
nc=diskinfo.fi_nclus;
ns=diskinfo.fi_sclus;
```

```
no=diskinfo.fi_bysec;

printf("Numar de unitati de alocare (clustere) :
%5u\n",nc);
printf("Numar de sectoare pe unitatea de alocare:
%5u\n",ns);
printf("Numar de octeti pe sector :
%5u\n",no);
dim = (long) nc*ns*no;
printf("\nNumar de octeti pe unitatea ");
if (u!=0) printf("%c:",0x40+u);else printf("C:");
printf( "%lu\n",dim);
printf("Identificator FAT :
%x\n",diskinfo.fi_fatid);
getch();
}
```

{FAT2.C - Obtinerea spatiului disponibil pe disc}

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

void main(void)
{
struct diskfree_t free;
unsigned int u;            //unitatea de disc
dorita
unsigned int nt;       //nr total de clustere
```

```
unsigned int nd;      //nr diponibil de clustere
unsigned int ns;      //nr de sectoare pe cluster
unsigned int no;      //nr de octeti pe sector
long unsigned int dim;        //dimensiune totala
disc
long unsigned int avail;  //dimensiune disc liber

clrscr();
printf("Obtinerea spatiului liber pe disc\n");
printf("Unitate de disc : (0-implicita,1-A:,2-B:3-
C:....) : ");
scanf("%d",&u);
if (_dos_getdiskfree(u, &free) != 0) {
printf("Eroare la apelul _dos_getdiskfree()\n");
exit(1);
  }

printf("\n");
nt=free.total_clusters;
nd=free.avail_clusters;
ns=free.bytes_per_sector;
no=free.sectors_per_cluster;

printf("Numar total de clustere :
%5u\n",nt);
printf("Numar de clustere libere :
%5u\n",nd);
printf("Numar de sectoare libere pe cluster :
%5u\n",ns);
```

```
printf("Numar de octeti liberi pe sector :
%5u\n",no);
dim = (long) nt*ns*no;
avail = (long) nd*ns*no;

printf("\nNumar total de octeti pe unitatea ");
if (u!=0) printf("%c:",0x40+u);else printf(" C :
");
printf( "%10lu\n",dim);
printf("\nNumar de octeti liberi pe unitatea ");
if (u!=0) printf("%c:",0x40+u);else printf("C :
");
printf( "%10lu\n",avail);
getch();
}
```

```
{FAT3.C - Obtinerea tipului dischetei}

#include <stdio.h>
#include <dos.h>

void main(void)
{
struct fatinfo diskinfo;
int flag = 0;

printf("Introduceti un disc in drive-ul A\n");
getchar();
```

```
getfat(1, &diskinfo);
/* obtine informatiile discului */
printf("\nDrive-ul A este: ");
switch((unsigned char) diskinfo.fi_fatid)
   {
case 0xF9:
printf("720K - Double Density\n");
break;

case 0xF0:
printf("1440K - High Density\n");
break;

default:
printf("Neformatat\n");
flag = 1;
   }

if (!flag)
   {
printf(" Numar de clustere   %5d\n",
diskinfo.fi_nclus);
printf(" Sectoare pe cluster %5d\n",
diskinfo.fi_sclus);
printf(" Octeti pe sector    %5d\n",
diskinfo.fi_bysec);
   }

getch();
```

```
}

{DRIVE.C - Determinarea drive-urilor logice
disponibile}

#include <stdio.h>
#include <conio.h>
#include <dir.h>

void main(void)
{
   int save, disk, disks;

   clrscr();
   /* Salvarea drive-ului original */
   save = getdisk();

   /* Afisarea numarului de drive-urilor logice */
   disks = setdisk(save);
   printf("Sistemul are %d drive-uri logice\n\n",
disks);

   /* Afisarea drive-urilor disponibile */
   printf("Drive-uri disponibile:\n");
   for (disk = 0;disk < 26;++disk)
   {
      setdisk(disk);
      if (disk == getdisk())
```

```
      printf("  drive-ul %c este disponibil\n",
disk + 'A');
   }
   setdisk(save);
   getch();
}
```

{DIR1.C - Crearea si stergerea un director}

```
#include <stdio.h>
#include <conio.h>
#include <process.h>
#include <dir.h>
#define DIRNAME "a:\\testdir.$$$"

void main(void)
{
   int stat;

   stat = mkdir(DIRNAME);
   if (!stat)
      printf("Director creat\n");
   else
   {
      printf("Nu se poate crea directorul\n");
      exit(1);
   }
   getch();
   system("dir/p a:");
```

```
    getch();
    stat = rmdir(DIRNAME);
    if (!stat)
        printf("\nDirector sters\n");
    else
    {
        perror("\nNu se poate sterge directorul\n");
        exit(1);
    }
    getch();
}
```

{DIR2.C - Schimbarea directorului}

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dir.h>

char old_dir[MAXDIR];
char new_dir[MAXDIR];

void main(void)
{
    clrscr();
    if (getcurdir(0, old_dir))
    {
        perror("getcurdir()");
        exit(1);
```

```
    }
    printf("Directorul curent este: \\%s\n",
old_dir);
    system("dir");
    getch();

    if (chdir("\\"))
    {
        perror("chdir()");
        exit(1);
    }

    if (getcurdir(0, new_dir))
    {
        perror("getcurdir()");
        exit(1);
    }
printf("\nDirectorul curent este acum:
          \\%s\n", new_dir);
    system("dir");
    getch();

    printf("\nRevenire la directorul original:
    \\%s\n", old_dir);
    if (chdir(old_dir))
    {
        perror("chdir()");
        exit(1);
    }
```

```
   system("dir");
   getch();
}
```

{FILE1.C - Crearea unui fisier si scrierea in el}

```
#include <dos.h>
#include <string.h>
#include <stdio.h>

#define n 100
int main(void)
{
  unsigned count;
  int handle;        // identificator de fisier
  char buf[n];       // buffer fisier
  char nume[8];      // nume fisier
  char nume0[n], *nume1="a:\\";

  /* citeste numele fisierului */
  clrscr();
  printf("Introduceti numele fisierului: ");
  strcpy(nume,nume1);
  scanf("%s",nume0);
  strcat(nume,nume0);

    /* creaza fisierul */
    if (_dos_creat(nume, _A_NORMAL, &handle) != 0)
// fisier normal
```

```
  {
    perror("Nu se poate crea fisierul !");
    return 1;
  }

  /* citeste datele */
  clrscr();
  printf("Introduceti datele: ");
  scanf("%s",buf);

  /* scrie in fisier */
if (_dos_write(handle, buf, strlen(buf),
     &count) != 0)
  {
    perror("Nu se poate scrie in fisier !");
    return 1;
  }

  /* inchide fisierul */
  _dos_close(handle);
  return 0;
}
```

{FILE2.C - Deschiderea unui fisier si citirea lungimii lui}

```
#include <dos.h>
#include <string.h>
#include <stdio.h>
```

```
#include <fcntl.h>

#define n 100
int main(void)
{
  unsigned count;
  int handle;          // identificator de fisier
  char buf[n];         // buffer fisier
  char nume[n];        // nume fisier
  char nume0[n], *nume1="a:\\";


  /* citeste numele fisierului */
  clrscr();
  printf("Introduceti numele fisierului: ");
  strcpy(nume,nume1);
  scanf("%s",nume0);
  strcat(nume,nume0);

  /* deschide fisierul */
  if (_dos_open(nume, O_RDWR, &handle) != 0) {
    perror("Nu se poate deschide fisierul !");
    return 1;
  }

  /* citeste din fisier */
  if (_dos_read(handle, buf, 10, &count) != 0) {
    perror("Nu se poate citi din fisier");
    return 1;
```

```
  }
  else
    printf("_dos_read: %d octeti cititi\n",count);
  return 0;

}
```

4.5. Desfăşurarea lucrării

a) Să se studieze tipurile de memorie secundara si functiile de management al directoarelor si fisierelor.

b) Să se studieze exemplele de la punctul 4.4. Să se verifice corectitudinea funcţionării programelor.

c) Să se ruleze utilitarele *Coretest*, *Dmi*, *Tmap* si *NDD*.

Tema 1: Sa se scrie un program care sa creeze un subdirector al directorului curent, sa schimbe directorul curent cu cel creat, sa creeze un fisier, sa scrie in fisier si sa il inchida.

Tema 2: Sa se scrie un program care sa schimbe directorul curent cu un subdirector dat, sa deschida un fisier din acest director, sa citeasca lungimea fisierului si sa il inchida.

Memoria cache. Gestiunea memoriei cache

Scopul lucrării

a) studiul caracteristicilor memoriei cache
b) studierea corespondenţei dintre memoria principală şi memoria cache (metode de mapare a memoriei cache).
c)analiza performanţelor diferitelor metode de mapare cu ajutorul programului de simulare *simcache.exe*.

1.Memoria cache

Memoria cache reprezintă un *buffer* de mare viteză plasat între procesor şi memoria principală ce are drept scop să stocheze porţiuni din memoria principală care sînt în utilizare curentă. De asemenea memoria cache poate fi plasată între memoria principală şi memoria secundară (virtuală).

Datorită faptului că memoria cache este cu un ordin de mărime mai rapidă decît memoria principală, se poate reduce timpul efectiv de acces la memorie cu condiţia ca aceasta să fie corect dimensionată şi să se utilizeze o politică adecvată de gestiune a memoriei cache.

Memoria cache este o memorie adresabilă prin conţinut şi este constituită dintrun index numit *cache directory* (CD) şi o *memorie RAM* (cu acces aleator).

Directorul CD constă în cîmpuri (tags) de adrese de blocuri de memorie şi biţi de control (de exemplu de protecţie). Cîmpurile de adrese "tags" conţin adresele blocurilor ce sînt la momentul de timp curent în memoria cache.

Adresa virtuală este translatată într-o adresă fizică de către sistemul de gestiune a memoriei.

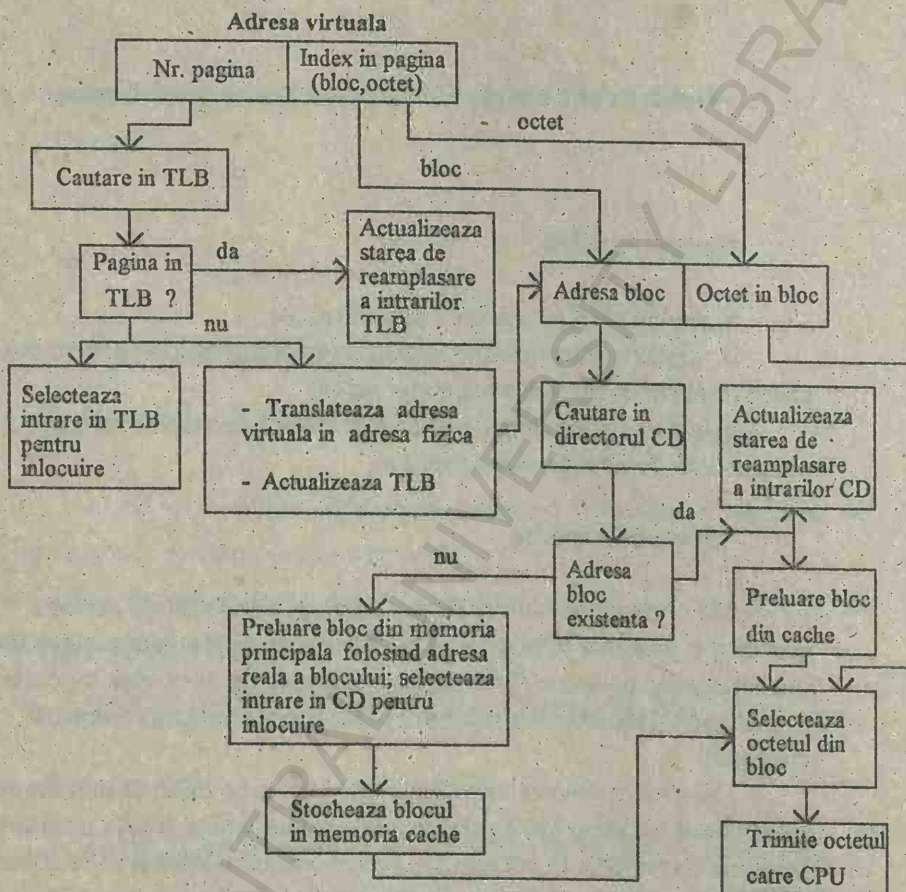


Figura 1. Operaţiile cu memoria cache (fetch din cache)

Există o tabelă de corespondenţă între adresa virtuală şi adresa fizică (*Translation Lookahead Buffer* - TLB) care conţine un număr de adrese virtuale (şi adresele fizice corespondente) cele mai recent utilizate.

Operaţia de access la memoria cache este prezentată în figura 1.

2. Organizarea memoriei cache

Există trei metode de bază pentru plasarea blocurilor din memoria principală în memoria cache (denumite asocieri sau mapări) :

- mapare directă
- mapare asociativă (*fully associative*)
- mapare asociativă pe seturi (*set associative*) .

Pentru simplificarea figurilor ce ilustrează organizarea memoriei cache (în aceste patru cazuri) se consideră dimensiunea memoriei cache de 2k cuvinte şi dimensiunea blocului de 16 cuvinte.

Memoria principală se alege de 256k cuvinte.

Adresa fizică va fi reprezentată pe 18 biţi ($256k = 2^8 . 2^{10} = 2^{18}$) . Exista $256k / 16 = 2^{18}/2^4 = 2^{14} = 16384$ blocuri în memoria principală şi $2k/16 = 2.2^{10} /2^4 = 128$ blocuri în memoria cache.

Maparea directă

Maparea directă a memoriei cache este ilustrată în figura 2.

Adresa din memoria principală (adresa fizică) se compune din cîmpurile . *tag*, *bloc* şi *cuvînt*.

În acest caz blocul i din memoria principală este asociat cu blocul (i mod nr_blocuri_în _cache) = (i mod 128) din memoria cache. Fiecare bloc din cache are asociat un cîmp "tag" (primii 7 biţi mai semnificativi ai adresei fizice în memoria principală).

Cîmpul tag din adresa memoriei principale se compară cu fiecare cîmp "tag" al memriei cach. După găsirea corespondenţei dintre cîmpurile tag se alege cîmpul "bloc" din adresa memoriei principale şi cîmpul "cuvînt" din aceeaşi adresă pentru selectarea octetului ce va fi transferat din blocul memoriei cache (dat de tag) în blocul memoriei principale (dat de cîmpul bloc şi cîmpul cuvînt al adresei în memoria principală).

Avantajul acestei mapări directe este accesul simultan la datele dorite şi la cîmpul tag (nu se face o căutare asupra cîmpului tag). Dacă cîmpul tag nu există în memoria cache , datele vor fi suprimate.

Un dezavantaj al mapării directe apare atunci cînd două sau mai multe blocuri din memoria principală, utilizate alternativ, sînt asociate

aceluiaşi bloc din cache. Această situaţie are probabilitate mică în sistemele uniprocesor, dar probabilitatea creşte pentru sistemele multiprocesor cu memorie cache comună.

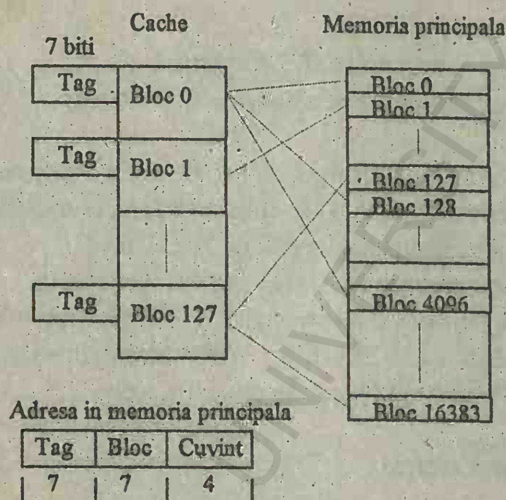


Figura 2. Organizarea cu mapare directă a memoriei cache

Maparea asociativă (fully associative)

Maparea asociativă a memoriei cache este ilustrată în figura 3.
Adresa din memoria principală (adresa fizică) se compune din cîmpurile : *tag* şi *cuvînt*.

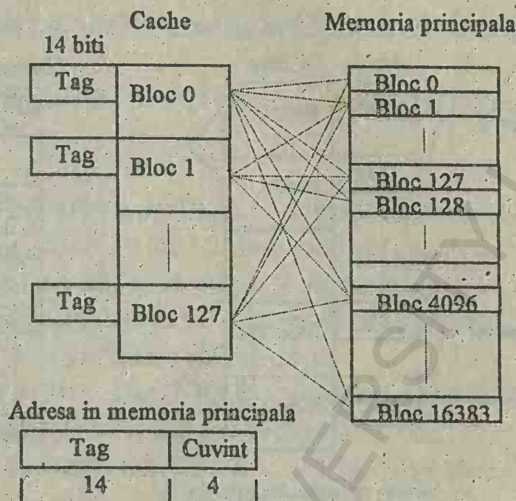


Figura 3. Organizarea cu mapare asociativă a memoriei cache

În acest caz *orice* bloc din memoria principală poate fi asociat cu *orice* bloc din memoria cache.

Metoda este cea mai bună, dar şi cea mai scumpă.

Maparea asociativă elimină conflictele de acces între blocuri, dar timpul de acces creşte datorită faptului că este necesară o căutare asociativă.

Maparea asociativă pe seturi

Maparea asociativă pe seturi a memoriei cache este ilustrată în figura 4.

Adresa din memoria principală (adresa fizică) se compune din cîmpurile : *tag*, *set* şi *cuvînt*.

Această mapare reprezintă un compromis între maparea directă şi maparea asociativă. Memoria cache este împărţită în S seturi . Blocul i , din memoria principală, poate corespunde cu orice bloc din memoria cache aparţinînd setului (i mod S).

Numărul de seturi , S, determină costul căutării (specifice mapării asociative). Setul se determină direct din adresă, apoi se caută cîmpul tag corespunzător adresei în set.

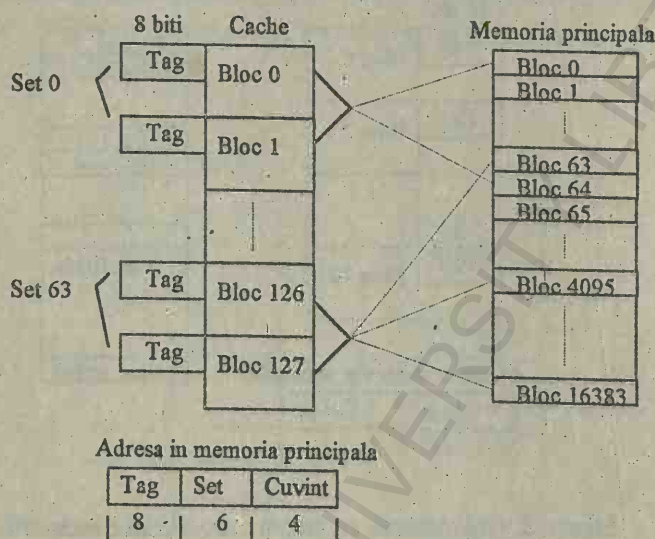


Figura 4. Organizarea cu mapare asociativă pe seturi a memoriei cache

Observaţii

- dimensiunea blocului în memoria cache este aleasă în raport cu proprietatea de *localizare* a programelor .
- memoria cache este adresată cu adrese fizice şi nu cu adrese virtuale (se elimină etapa de translatare a adresei virtuale în adresă fizică)
- algoritmii de reamplasare a blocurilor din memoria cache determină performanţa memoriei cache, exprimată ca *rată de pierderi* (număr de încercări eşuate de a găsi informaţia dorită în cache).

3.Politici de reamplasarea blocurilor din memoria cache

Există următoarele politici de reamplasare a blocurilor din memoria cache:

1. **Least recently used (LRU)** - se înlocuieşte blocul din cache cel mai puţin recent utilizat , relativ la momentul de timp curent (se face o analiza a utilizarii blocurilor existente în memoria cache cu un anumit număr de momente de timp în urmă în raport cu momentul de timp curent, la care trebuie să se producă o reamplasare).
2. **First In First Out (FIFO)** - se înlocuieşte blocul din cache care a stat în memoria cache cel mai mare interval de timp.
3. **Last In First Out (LIFO)** - se înlocuieşte blocul din cache care a stat în memoria cache cel mai mic interval de timp.
4. **Random (RAND)** - se alege, în mod aleator ,un bloc din memoria cache ce va fi înlocuit.

4. Programul de simulare a metodelor de mapare a memoriei cache

Pentru evaluarea performanţelor diferitelor metode de mapare a memoriei cache se va utiliza programul de simulare *simcache*, care ilustrează grafic *rata de pierderi* calcualtă ca raport între numărul de încercări eşuate de găsire a informaţiei dorite în cache şi numărul total de accese la memoria cache.

Programul simcache are următoarea organigramă generală (figura 5) :

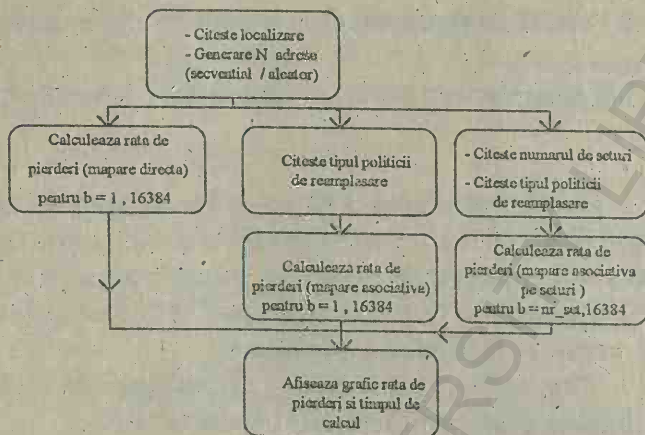


Figura 5. Organigrama programului SIMCACHE

Pentru o sesiune de lucru cu programul *simcache* trebuie date urmatoarele comenzi :

SIMULATOR MEMORIE CACHE

Generare adrese aleator/secvential [a/s] :s

Gradul de localizare [0 : fara localizare] :100

START SIMULARE MAPARE DIRECTA: ______________ END

Numar de seturi (exprimat in biti) :3 <—— nr. seturi = 2^k

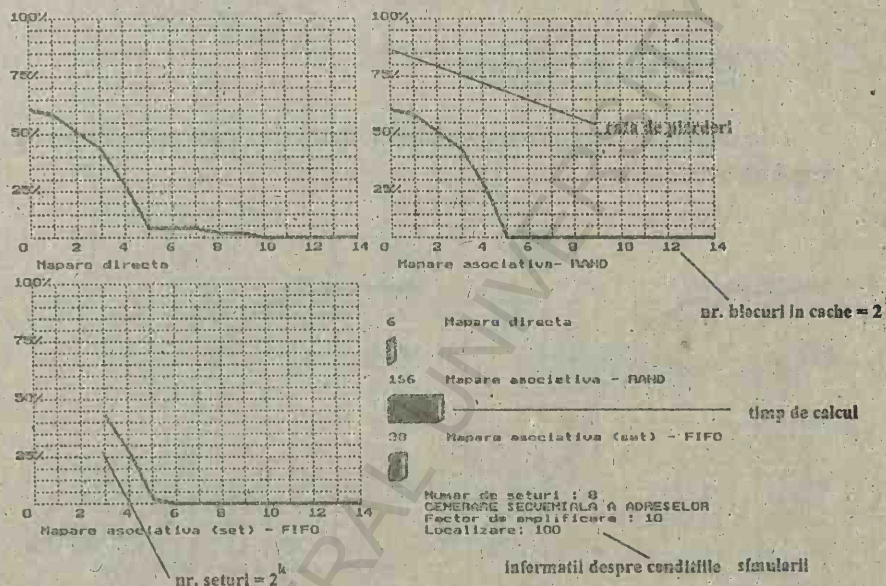
Politici de reamplasare : 0->RAND 1->LRU 2->FIFO 3->LIFO

Politica de reamplasare mapare asociativa :0
Politica de reamplasare mapare asociativa pe seturi :2

START SIMULARE MAPARE ASOCIATIVA: ______________ END

START SIMULARE MAPARE ASOCIATIVA PE SETURI:
_______________ END

Factor de amplificare a afisarii :10



Modificare factor de amplificare ? [d/n] : n

Iesire din simulator ? [d/n] : d

Gradul de localizare indică gama de generare a adreselor (diferenţa între adresa maximă şi minimă generate) în multiplii de 16 blocuri . Pentru grad de localizare nul adresele se generează pe o gamă foarte mare (pe 28 biţi).

Adresele se pot genera aleator pe întreg intervalul de 28 biti sau pe un interval dat de gradul de localizare.

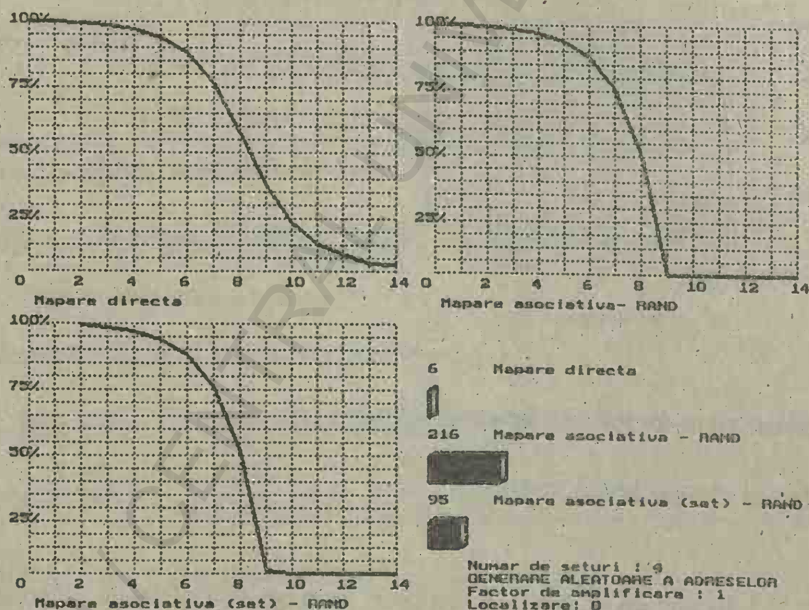
Generarea secvenţială a adreselor presupune că se generează aleator o adresă de start şi o adresă de subrutină după care se generează un număr

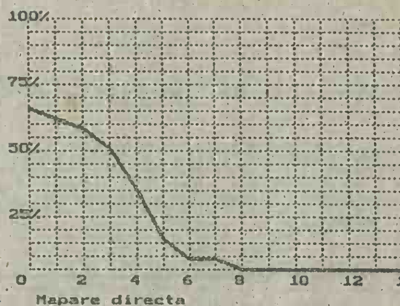
de adrese la rînd (secvenţial) apoi se simulează saltul la subrutină şi revenirea din subrutină. Toate adresele au valori corespunzătoare gradului de localizare.

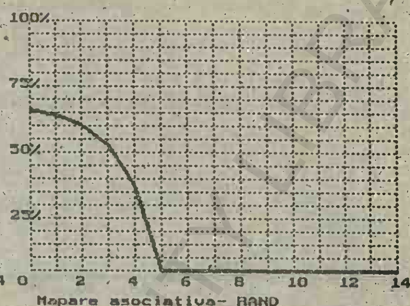
Timpul de calcul este măsurat prin întreruperi periodice generate de ceasul de timp real al PC -ului.

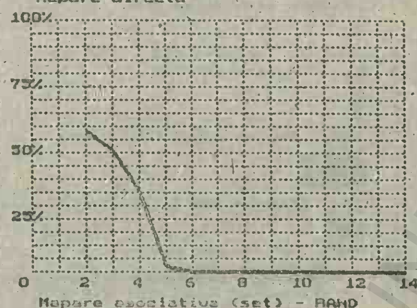
5. Exemple de rezultate

În continuare sînt prezentate cîteva dintre rezultatele simulărilor cu programul *simcache*:







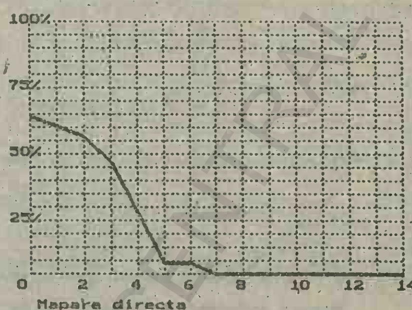


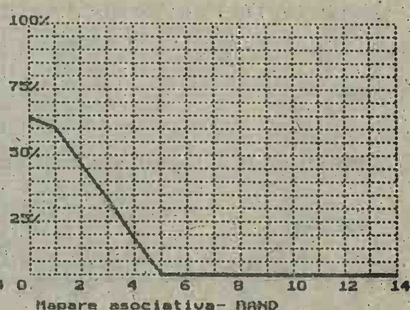
6 Mapare directa

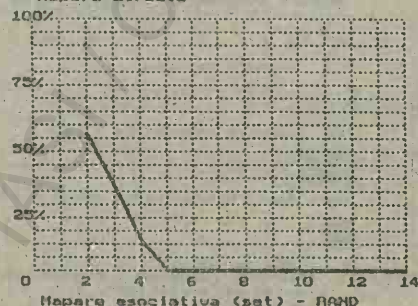
166 Mapare asociativa - RAND

73 Mapare asociativa (set) - RAND

Numar de seturi : 4
GENERARE SECVENIALA A ADRESELOR
Factor de amplificare : 10
Localizare: 100





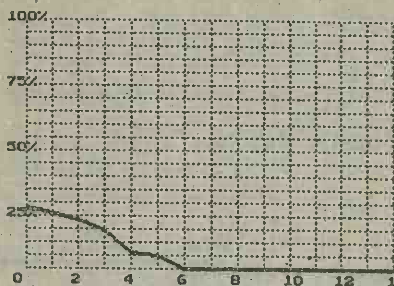


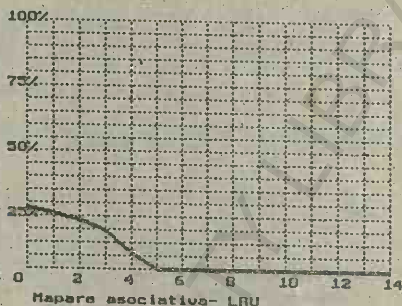
6 Mapare directa

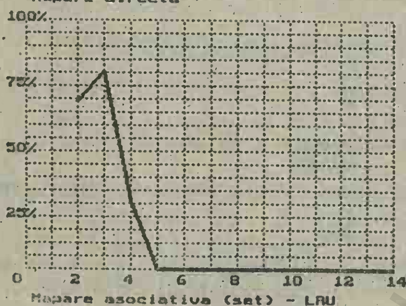
164 Mapare asociativa - RAND

74 Mapare asociativa (set) - RAND

Numar de seturi : 4
GENERARE SECVENIALA A ADRESELOR
Factor de amplificare : 10
Localizare: 10





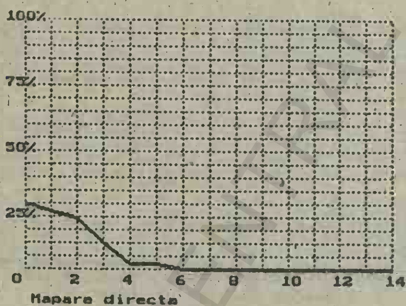


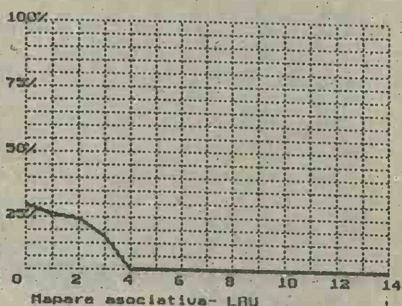
5 Mapare directa

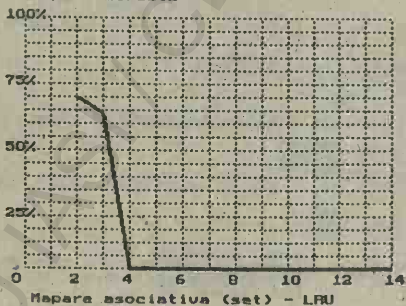
163 Mapare asociativa - LRU

73 Mapare asociativa (set) - LRU

Numar de seturi : 4
GENERARE SECVENIALA A ADRESELOR
Factor de amplificare : 5
Localizare: 100







7 Mapare directa

162 Mapare asociativa - LRU

74 Mapare asociativa (set) - LRU

Numar de seturi : 4
GENERARE SECVENIALA A ADRESELOR
Factor de amplificare : 5
Localizare: 10

6. Desfăşurarea lucrării

a) Să se studieze metodele de mapare descrise.
b) Să se ruleze programul de simulare *simcache* după următorul tabel:

Metoda mapare	Generare adresa	Localizare	Nr. seturi	Politica reamplasare	Timp de calcul	**Observatii referitoare rata de pierderi**
directa	aleator	0	-	-		
directa	secvential	10	-	-		
directa	secvential	100	-	-		
directa	secvential	1000	-	-		
asociativa	aleator	0	-	RAND		
asociativa	aleator	0	-	FIFO		
asociativa	aleator	0	-	LIFO		
asociativa	aleator	0	-	LRU		
asociativa	secvential	100	-	RAND		
asociativa	secvential	100	-	FIFO		
asociativa	secvential	100	-	LIFO		
asociativa	secvential	100	-	LRU		
asociativa set	aleator	0	1,2,4,8,16	RAND		
asociativa set	aleator	0	1,2,4,8,16	FIFO		
asociativa set	aleator	0	1,2,4,8,16	LIFO		
asociativa set	aleator	0	1,2,4,8,16	LRU		

asociativa set	secvential	100	1,2,4,8,16	RAND		
asociativa set	secvential	100	1,2,4,8,16	FIFO		
asociativa set	secvential	100	1,2,4,8,16	LIFO		
asociativa set	secvential	100	1,2,4,8,16	LRU		

Temă

a) Să se discute avantajele şi dezavantajele diferitelor metode de mapare din punctul de vedere al ratei de pierderi şi a timpului de calcul.

b) Care este influenţa gradului de localizare asupra performanţelor memoriei cache ?

c) Care este influenţa numărului de seturi asupra performanţelor memoriei cache ?

Moduri de lucru cu monitoarele video ale calculatoarelor personale

Scopul lucrării

a) Studiul modului de lucru text.

b) Studiul modului de lucru grafic.

c) Utilizarea funcţiilor de nivel înalt pentru controlul modurilor de lucru ale monitoarelor video.

6.1. Introducere

Pentru un calculator personal (PC) există două moduri diferite de afişare a informaţiilor pe un *ecran*: mod *text* (caractere alfabetice, numerice, de punctuaţie şi speciale) şi mod *grafic*. Toate dispozitivele de afişare, numite *monitoare* sau *display*-uri utilizează tehnica de compunere a imaginilor text sau grafice prin "aprinderea" pe ecran cu intensităţi şi culori diferite a unor zone de dimensiuni foarte reduse, aproape punctiforme, numite *pixeli*, organizate într-o reţea de puncte.

Densitatea acestei reţele de puncte ale ecranului, numită *rezoluţie*, constituie o caracteristică importantă a echipamentului, considerat cu atât mai bun cu cât rezoluţia este mai mare. Alte caracteristici care le diferenţiază sunt: numărul de culori, viteza de afişare, numărul de puncte alocate zonei de afişare a unui caracter, etc. Interfaţarea monitoarelor cu unitatea centrală este realizată prin intermediul unor dispozitive numite

adaptoare, cum ar fi: CGA (Color Grafics Adapter), EGA (Extended Grafics Adapter), VGA (Video Grafics Adapter).

6.2. Modul grafic

În mod grafic, monitorul unui PC lucrează ca un televizor, afişând o imagine formată dintr-un număr mare de puncte independente, numite elemente de imagine sau *pixeli*. Culoarea şi luminozitatea fiecărui pixel pot fi stabilite independent de culoarea şi luminozitatea celorlalţi pixeli.

Imaginea afişată de monitorul video este formată dintr-un număr oarecare de linii orizontale, fiecare linie având un număr oarecare de pixeli. Produsul acestora reprezintă numărul total de pixeli sau *rezoluţia* ecranului. *Rezoluţiile tipice* ale ecranelor video în mod grafic sunt:

adaptor (*driver*)	moduri grafice (*graphics_modes*)	cod	coloane x linii	paletă culori	pagini
CGA	CGAC0	0	320 x 200	C0	1
	CGAC1	1	320 x 200	C1	1
	CGAC2	2	320 x 200	C2	1
	CGAC3	3	320/x 200	C3	1
	CGAHI	4	640 x 200	2 culori	1
MCGA	MCGAC0	0	320 x 200	C0	1
	MCGAC1	1	320 x 200	C1	1
	MCGAC2	2	320 x 200	C2	1
	MCGAC3	3	320 x 200	C3	1
	MCGAMED	4	640 x 200	2 culori	1
	MCGAHI	5	640 x 480	2 culori	1
EGA	EGALO	0	640 x 200	16 culori	4
	EGAHI	1	640 x 350	16 culori	2
EGA64	EGA64LO	0	640 x 200	16 culori	1
	EGA64HI	1	640 x 350	4 culori	1
EGA-MONO	EGAMONOHI	3	640 x 350	2 culori	1
	EGAMONOHI	3	640 x 350	2 culori	2
HERC	HERCMONOHI	0	720 x 348	2 culori	2
ATT400	ATT400C0	0	320 x 200	C0	1

	ATT400C1	1	320 x 200	C1	1
	ATT400C2	2	320 x 200	C2	1
	ATT400C3	3	320 x 200	C3	1
	ATT400MED	4	640 x 200	2 culori	1
	ATT400HI	5	640 x 400	2 culori	1
VGA	VGALO	0	640 x 200	16 culori	2
	VGAMED	1	640 x 350	16 culori	2
	VGAHI	2	640 x 480	16 culori	1
PC3270	PC3270HI	0	720 x 350	2 culori	1
IBM8514	IBM8514HI	1	1024 x 760	256 culori	
	IBM8514LO	0	640 x 480	256 culori	

6.3. Modul text

În mod text, caracterele sunt afişate pe *linii*, de la stânga la dreapta ecranului şi de sus în jos. Poziţia curentă de afişare este indicată de un *cursor*, care se mută automat, odată cu afişarea caracterelor. Când cursorul atinge ultima poziţie a ecranului (ultimul caracter al ultimului rând) întregul text afişat pe ecran se mută în sus cu o poziţie, pierzându-se textul aflat pe prima linie şi creându-se spaţiu pentru o linie nouă, la baza ecranului. Cursorul este poziţionat în stânga noii linii.

În mod text, pentru fiecare caracter se păstrează în memorie două informaţii: *codul caracterului* care se afişează, şi *culorile* utilizate pentru desenarea *caracterului* şi respectiv a *fundalului* pe care se face afişarea. Structura octetului care codifică culorile de afişare este:

B	F	F	F	C	C	C	C
7	6	5	4	3	2	1	0

unde B = mod afişare (0 = continuă, 1 = intermitentă);

FFF = codul culorii utilizate pentru fundal;

CCCC = codul culorii utilizate pentru afişarea caracterului.

În afară de caracterele ASCII "obişnuite": litere, cifre, semne de punctuaţie, pentru afişarea pe ecran se pot utiliza şi caractere speciale din setul ASCII extins. Caracterele care fac parte din acest set şi nu fac parte din cel "clasic" sunt cele *semigrafice*, cu care se pot face diferite desene, caractere din alfabetul grecesc şi caractere cu specific matematic.

Pentru a indica poziţia curentă, pe ecran se afişează un *cursor* de formă dreptunghiulară, a cărui dimensiune în cadrul unui caracter poate fi controlată prin program.

Pentru a memora conţinutul ecranului se utilizează o zonă de *memorie* specială în afara spaţiului de memorie utilizat pentru programe, denumită *memorie ecran*. Numărul maxim de puncte ce pot fi afişate şi numărul de culori utilizate pentru aceasta, în modul grafic determină dimensiunea memoriei ecran. Pentru o interfaţă clasică memoria utilizată pentru ecran are adresa de segment *0xB800*.

Deoarece pe ecran exstă mult mai multe puncte decât caractere, în memoria ecran pot fi memorate informaţiile corespunzătoare mai multor imagini de ecran text. De exemplu, pentru un ecran text în rezoluţie mare sunt necesari 80 x 25 x 2 = 4000 octeţi. În cazul unei interfeţe standard, memoria ecran are 16 kocteţi, în această memorie se pot păstra informaţiile pentru patru ecrane de tip text. Dintre aceste ecrane la un moment dat este activ (selectat) unul singur.

Adresa ocupată în memoria ecran de către informaţia corespunzătoare unui caracter este:

adr_car = (linie * 80 + coloana) * 2 + nr_ecran * 4096

Rezoluţiile tipice ale ecranelor video în mod text sunt:

- CGA: 25 de rânduri a 80 sau 40 de coloane de caractere în matrici de 8x8 pixeli, 16 culori;

- EGA: 25 sau 43 de rânduri a 80 de coloane de caractere în matrici de 7x16 pixeli, 16 culori;
- VGA: 25 sau 50 de rânduri a 80 de coloane de caractere în matrici de 9x16 puncte, 16 culori.

6.4. Exemple de funcţii C pentru controlul monitorului video

6.4.1. Configurarea sistemului grafic

detectgraph : Determinarea driver-ului şi a modului grafic utilizate verificând hardware-ul

Declaraţie: void far detectgraph(int far *graphdriver, int far *graphmode);

Parametru Semnificatie

*graphdriver intreg care specifică driver-ul grafic utilizat (constanţă din enum. *graphics_drivers*)

*graphmode specifică modul grafic initial (valoare din enum *graphics_modes*)

Observaţie: *detectgraph* detectează adaptorul grafic al sistemului şi alege modul care oferă cea mai mare rezolutie pentru adaptor.

initgraph : inițializează sistemul grafic

Declarație: void far initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver);

Parametru *Semnificatie*

*graphdriver intreg care specifică driver-ul grafic utilizat (constantă din enum. *graphics_drivers*)

*graphmode specifică modul grafic initial (valoare din enum *graphics_modes*)

pathtodriver specifică calea către directorul în care *initgraph* caută driver-ele grafice (*.BGI)

- mai întâi (dacă nu sunt acolo, *initgraph* caută în directorul curent); dacă *pathtodriver*
- este nulă, fișierele driver trebuie să fie în directorul curent

Observații:

Pentru a utiliza sistemul grafic, trebuie mai întâi apelat *initgraph*.

- *initgraph* <u>inițializează</u> sistemul grafic încărcând driver-ul grafic de pe disc (sau validând un driver înregistrat) trecând astfel sistemul în mod grafic.
- *initgraph* <u>resetează</u> toate setările grafice (culoare, paletă, poziție curentă, etc.) la valoarea lor implicită și resetează *graphresult* la 0. După apelul *initgraph*, **graphdriver* este setat la driver-ul grafic curent, iar **graphmode* este setat la modul grafic curent.

Se poate cere *initgraph* să utilizeze un driver şi mod grafic particular, sau să autodetecteze adaptorul video ataşat curent.

Dacă se cere *initgraph* să autodetecteze (**graphdriver = DETECT*), el apelează *detectgraph* pentru a selecta driver-ul şi modul grafic la cea mai mare rezoluţie disponibilă pentru driver-ul detectat.

graphics_drivers : enumerare

constantă	valoare	constantă	valoare	constantă	valoare
DETECT	0 (cere autodetecţie)	EGA64	4	ATT400	8
CGA	1	EGAMONO	5	VGA	9
MCGA	2	IBM8514	6	PC3270	10
EGA	3	HERCMONO	7		

closegraph : închide sistemul grafic

Declaraţie: void far closegraph(void);

Observaţie: *closegraph* <u>eliberează</u> toată memoria alocată pentru sistemul grafic, apoi reface ecranul în modul anterior apelului *initgraph.*

setviewport : Set-ează fereastra grafică curentă

Declaratie: void far setviewport(int left, int top, int right, int bottom, int clip);

Observaţii: (*left,top*) este colţul din stânga-sus, iar (*right,bottom*) este colţul din dreapta-jos al ferestrei. Poziţia curentă (CP) este mutată în (0,0) al noii ferestre

setactivepage : Set-ează pagina activă

setvisualpage : Set-ează pagina vizibilă

Declarații: void far setactivepage(int page);

void far setvisualpage(int page);

getcolor : returnează culoarea curentă pentru desenare

setcolor : set-ează culoarea curentă pentru desenare

Declarații: int far getcolor(void);

void far setcolor(int color);

getbkcolor : returnează culoarea curentă pentru fond

setbkcolor : set-ează culoarea curentă pentru fond

Declarații: int far getbkcolor(void);

void far setbkcolor(int color);

CGA_COLORS : enumerare

Nume paletă	Constantă asociată 1	2	3
CGA0	CGA_LIGHTGREEN	CGA_LIGHTRED	CGA_YELLOW
CGA1	CGA_LIGHTCYAN	CGA_LIGHT MAGENTA	CGA_WHITE
CGA2	CGA_GREEN	CGA_RED	CGA_BROWN
CGA3	CGA_CYAN	CGA_MAGENTA	CGA_LIGHTGRAY

EGA_ COLORS : enumerare

Constantă	Valoare	Constantă	Valoare
EGA_BLACK	0	EGA_DARKGRAY	56
EGA_BLUE	1	EGA_LIGHTBLUE	57
EGA_GREEN	2	EGA_LIGHTGREEN	58
EGA_CYAN	3	EGA_LIGHTCYAN	59
EGA_RED	4	EGA_LIGHTRED	60
EGA_MAGENTA	5	EGA_LIGHTMAGENTA	61
EGA_LIGHTGRAY	7	EGA_YELLOW	62
EGA_BROWN	20	EGA_WHITE	63

settextstyle : set-ează caracteristicile curente ale textului

Declaratie: void far settextstyle(int font, int direction, int charsize);

6.4.2. Utilizarea sistemului grafic

cleardevice : şterge ecranul grafic

Declarație: void far cleardevice(void);

Observații: ştergerea constă în acoperirea cu culoarea curentă a fundalului

cleardevice şterge întregul ecranul grafic şi muta CP (poziția punctului curent) în origine (0,0).

imagesize : returnează numărul de octeți necesari pentru stocarea unei imagini

Declarație: unsigned far imagesize(int left, int top, int right, int bottom);

getimage : salvează o imagine dintr-o regiune specificată în memorie

putimage : pune o imagine pe ecran

Declarații: void far getimage(int left, int top, int right, int bottom, void far *bitmap);

void far putimage(int left, int top, void far *bitmap, int op);

Observații: *putimage* repune o imagine anterior salvată cu *getimage* pe ecran, cu colțul din stînga sus în (left,top).

getimage copiază o imagine de pe ecran în memorie.

Parametru	*Semnificatie*
bitmap	adresa unei zone din memorie în care este salvată imaginea; primele două cuvinte ale zonei sunt utilizate pentru lățimea și înălțimea dreptunghiului
op	specifică un operator de combinare care controlează modul de calcul al culorii pentru fiecare pixel destinație pe ecran, bazîndu-se pe pixel-ul aflat deja pe ecran și sursa corespunzătoare din memorie

putimage_ops: enumerare, dă numele operatorilor de combinare pentru *putimage*.

Constantă	Valoare	Funcție
COPY_PUT	0	Copiază imaginea sursă pe ecran
XOR_PUT	1	OR eXclusiv între imaginea sursă și cea aflată deja pe ecran
OR_PUT	2	OR între imaginea sursă și cea aflată deja pe

		ecran
AND_PUT	3	AND între imaginea sursă şi cea aflată deja pe ecran
NOT_PUT	4	Copiază inversul imaginii sursă pe ecran

getmaxx : returnează coordonata x (ordonata) maximă a ecranului

getmaxy : returnează coordonata y (abscisa) maximă a ecranului

Declaraţii: int far getmaxx(void);

int far getmaxy(void);

getx : returnează poziţia curentă a coordonatei x

gety : returnează poziţia curentă a coordonatei y

Declaraţii: int far getx(void);

int far gety(void);

moverel : mută poziţia curentă (CP) la o distanţă relativă

moveto : mută CP în (x, y)

Declaraţii: void far moverel(int dx, int dy);

void far moveto(int x, int y);

getpixel : obţine culoarea unui pixel specificat (x,y)

putpixel : desenează un pixel într-un punct specificat (x,y)

Declaraţii: unsigned far getpixel(int x, int y);

void far putpixel(int x, int y, int color);

line : desenează o linie între două puncte specificate

linerel : desenează o linie la distanţă relativă faţă de poziţia curentă (CP)

lineto : desenează o linie din CP la (x,y)

Declaraţii: void far line(int x1, int y1, int x2, int y2);

void far linerel(int dx, int dy);

void far lineto(int x, int y);

Observaţii: *line* desenează o linie de la (x1, y1) la (x2, y2) utilizând culoarea, stilul liniei, şi grosimea curente fără a modifica poziţia punctului curent (CP).

linerel desenează o linie de la CP la un punct care este la o distanţă relativă (dx, dy) faţă de CP, apoi avansează CP cu (dx, dy).

lineto desenează o linie de la CP la (x, y), apoi mută CP în (x, y).

rectangle : desenează un dreptunghi (în mod grafic)

Declaraţie: void far rectangle(int left, int top, int right, int bottom);

Observaţii: *rectangle* desenează un dreptunghi cu culoarea, stilul liniei, şi grosimea curente.

(*left,top*) este colţul din stânga-sus al dreptunghiului, iar (*right,bottom*) este colţul din dreapta-jos al dreptunghiului.

arc : desenează un arc de cerc

circle : desenează un cerc

pieslice : desenează şi umple un sector de cerc

Declarații: void far arc(int x, int y, int stangle, int endangle, int radius);
void far circle(int x, int y, int radius);
void far pieslice(int x, int y, int stangle, int endangle, int radius);

Parametru	*Semnificatie*
(x,y)	Centrul cercului
stangle	Unghiul inițial în grade
endangle	Unghiul final în grade
radius	Raza cercului

outtext : afişează un şir de caractere într-o fereastră video
outtextxy : afişează un şir de caractere într-o locaţie specificată (x,y)

Declarații: void far outtext(char far *textstring);
void far outtextxy(int x, int y, char far *textstring);

6.4.3. Configurarea modului text

textattr : set-ează atributele textului pentru ferestre de text
textbackground : selectează o nouă culoare a fondului textului
textcolor : selectează o nouă culoare a caracterului

Declarații: void textattr(int newattr);
void textbackground(int newcolor);
void textcolor(int newcolor);

Parametru	*Semnificatie*
newattr	Informaţiile culorilor codate
Exemplu:	textcolor(CYAN + BLINK);

COLORS: enumerare

Constantă	Valoare	Utilizată ca fond	Constantă	Valoare	Utilizată ca fond
BLACK	0	Da	DARKGRAY	8	Nu
BLUE	1	Da	LIGHTBLUE	9	Nu
GREEN	2	Da	LIGHTGREEN	10	Nu
CYAN	3	Da	LIGHTCYAN	11	Nu
RED	4	Da	LIGHTRED	12	Nu
MAGENTA	5	Da	LIGHTMAGENTA	13	Nu
BROWN	6	Da	YELLOW	14	Nu
LIGHTGRAY	7	Da	WHITE	15	Nu
BLINK	128	Nu ***			

*** Pentru a afişa caractere intermitent în mod text, trebuie adunat BLINK la culoarea fondului.

6.4.4. Utilizarea modului text

window : Defineşte fereastra activă în mod text

Declaraţie: void window(int left, int top, int right, int bottom);

wherex : returnează poziţia curentă pe orizontală a cursorului în fereastra text curentă

wherey : returnează poziţia curentă pe verticală a cursorului în fereastra text curentă

Declarații: int wherex(void);

int wherey(void);

gotoxy : Poziționează cursorul în fereastra text

Declarație: void gotoxy(int x, int y);

gettext : copiază un text de pe ecran în mod text în memorie

puttext : copiază un text din memorie pe ecran în mod text

Declarații: int gettext(int left, int top, int right, int bottom, void*destin);

int puttext(int left, int top, int right, int bottom, void*source);

Observații: *gettext* memorează conținutul ecranului din dreptunghiul definit de (left, top) și (right, bottom) în zona de memorie *destin. *puttext* scrie conținutul zonei de memorie *source pe ecran.

gettextinfo : Obține informațiile modului text

Declarație: void gettextinfo(struct text_info *r);

```
struct text_info {
    unsigned char winleft;    /* coordonata din stânga a ferestrei */
    unsigned char wintop;     /* coordonata de sus a ferestrei */
    unsigned char winright;   /* coordonata din dreapta a ferestrei */
    unsigned char winbottom;        /* coordonata de jos a ferestrei */
    unsigned char attribute;        /* atributele textului */
    unsigned char normattr;         /* atributele normale */
```

unsigned char currmode; /* modul video curent: BW40, BW80, C40, C80, or C4350 */
unsigned char screenheight; /* înălţimea textului */
unsigned char screenwidth; /* lăţimea textului */
unsigned char curx; /* coordonata x în fereastra curentă */
unsigned char cury; /* coordonata z în fereastra curentă */ };

6.5. Exemple de programe

```
{DETECTGR.C - Detectarea drive-ului si modului
grafic}

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

/* numele diferitelor placi video suportate */
char *dname[] = { "autodetectat", "CGA", "MCGA",
"EGA", "64K EGA",
"EGA monocrom", "IBM 8514",     "Hercules
monocrom",    "AT&T 6300 PC", "VGA",          "IBM
3270 PC"     };

void main(void)
{
   /* returneaza informatia privind hardware-ul
detectat */
   int gdriver, gmode, errorcode;
  /* detecteaza hardware-ul grafic disponibil */
   detectgraph(&gdriver, &gmode);
   /* citeste rezultatul apelului detectgraph */
   errorcode = graphresult();
   if (errorcode != grOk)  /* a aparut o eroare */
   {
```

```
      printf("Eroare grafica : %s\n",
grapherrormsg(errorcode));
      printf(":");
      getch();
      exit(1); /* terminat cu cod de eroare */
   }
   /* afiseaza informatia detectata */
   clrscr();
   printf("Aveti o placa video %s.\n",
dname[gdriver]);
   printf("Apasati orice tasta pentru oprire:");
   getch();
}
```

{GETIMAGE.C - Salveaza si reface continutul ecranului grafic}

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
int maxx, maxy;

void save_screen(void far *buf[4])
{
   unsigned size;
   int ystart=0, yend, yincr, block;
   yincr = (maxy+1) / 4;
```

```
   yend = yincr;
   size = imagesize(0, ystart, maxx, yend);
/* obtine dimensiunea in octeti a imaginii */
   for (block=0; block<=3; block++)
   {
      if ((buf[block] = farmalloc(size)) == NULL)
      {
         closegraph();
      printf("Eroare: nu este destul spatiu
heap.\n");
         exit(1);
      }
      getimage(0, ystart, maxx, yend, buf[block]);
      ystart = yend + 1;
      yend += yincr + 1;
   }
}
void restore_screen(void far *buf[4])
{
   int ystart=0, yend, yincr, block;
   yincr = (maxy+1) / 4;
   yend = yincr;
   for (block=0; block<=3; block++)
   {
      putimage(0, ystart, buf[block], COPY_PUT);
      farfree(buf[block]);
      ystart = yend + 1;
      yend += yincr + 1;
   }
}
```

```
void main(void)
{
   int gdriver=DETECT, gmode, errorcode;
   void far *ptr[4];
   initgraph(&gdriver, &gmode,
c:\\borlandc\\bgi");
   errorcode = graphresult();
   if (errorcode != grOk)
   {
      printf("Eroare grafica: %s\n",
grapherrormsg(errorcode));
      printf("Apasati orice tasta pentru
oprire:");
      getch();
      exit(1);
   }
   maxx = getmaxx();
   maxy = getmaxy();
   /* deseneaza o imagine pe ecran */
   rectangle(0, 0, maxx, maxy);
   line(0, 0, maxx, maxy);
   line(0, maxy, maxx, 0);
   save_screen(ptr);    /* salveaza ecranul curent
*/
   getch();             /* pauza */
   cleardevice();       /* clear screen */
   getch();             /* pauza */
   restore_screen(ptr); /* restore the screen */
   getch();             /* pauza */
   closegraph();
```

```
}
{IMAGSIZE.C - Salveaza si reface continutul
ecranului grafic}

#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#define ARROW_SIZE 10

void draw_arrow(int x, int y)
{
   /* deseneaza o sageata pe ecran */
   moveto(x, y);
   linerel(4*ARROW_SIZE, 0);
   linerel(-2*ARROW_SIZE, -1*ARROW_SIZE);
   linerel(0, 2*ARROW_SIZE);
   linerel(2*ARROW_SIZE, -1*ARROW_SIZE);
}
void main(void)
{
   int gdriver = DETECT, gmode, errorcode;
   void *arrow;
   int x, y, maxx;
   unsigned int size;

   initgraph(&gdriver, &gmode, "c:\\bc31\\bgi");
   errorcode = graphresult();
   if (errorcode != grOk)  /* a aparut o eroare */
   {
```

```
      printf("Eroare grafica: %s\n",
grapherrormsg(errorcode));
      printf("Apasa o tasta pentru oprire:");
      getch();
      exit(1); /* terminare cu un cod de eroare */
   }
   maxx = getmaxx();
   x = 0;
   y = getmaxy() / 2;
   /* deseneaza sageata */
   draw_arrow(x, y);
   /* calculeaza dimensiunea imaginii */
   size = imagesize(x, y-ARROW_SIZE,
x+(4*ARROW_SIZE), y+ARROW_SIZE);
   /* aloca memorie pentru a stoca imaginea */
   arrow = malloc(size);
   /* copiaza sageata */
   getimage(x, y-ARROW_SIZE, x+(4*ARROW_SIZE),
y+ARROW_SIZE, arrow);
   /* repeta pana cand o tasta e apasata */
   while (!kbhit())
   {
      /* sterge vechea sageata */
      delay(10);
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT);
      x += ARROW_SIZE;
      if (x >= maxx)
      x = 0;
      /* afiseaza noua sageata */
      putimage(x, y-ARROW_SIZE, arrow, XOR_PUT)
```

```
    }
    /* elibereaza memoria */
    free(arrow);
    closegraph();
}
```

{SETPAGE.C - Schimba pagina grafica activa si pagina grafica vizibila}

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>

void main(void)
{
  /* selecteaza un driver si un mode care suporta
pagini multiple */
  int gdriver = EGA, gmode = EGAHI, errorcode;
  int x, y, ht;

  initgraph(&gdriver, &gmode, "c:\\bc31\\bgi\\");
  errorcode = graphresult();
  if (errorcode != grOk)
    {
    printf("Eroare grafica: %s\n",
grapherrormsg(errorcode));
    printf("Apasa orice tasta pentru oprire:");
    getch();
```

```
    exit(1);
  }
  x = getmaxx() / 2;
  y = getmaxy() / 2;
  ht = textheight("W");

  /*  selecteaza pagina invizibila pentru desenare
(nr. 1) */
  setactivepage(1);
  /* deseneaza o linie in pagina nr. 1 */
  line(0, 0, getmaxx(), getmaxy());
  /* scrie un mesaj in pagina nr. 1 */
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  outtextxy(x, y, "Aceasta este pagina nr. 1:");
  outtextxy(x, y+ht, "Apasa orice tasta pentru
oprire:");

  /* selecteaza desenare in pagina nr. 0 */
  setactivepage(0);
  /* scrie un mesaj in pagina nr. 0 */
  outtextxy(x, y, "Aceasta este pagina nr. 0.");
  outtextxy(x, y+ht, "Apasa orice tasta pentru a
vedea pagina nr. 1:");
  getch();

  /* selecteaza pagina nr. 1 ca pagina visibila */
  setvisualpage(1);
  getch();
  closegraph();
}
```

{TEXTATTR.C - Modifica atributele textului}

```
#include <conio.h>
void main(void)
{
   int i,j;
   for (i=0; i<8; i++)
   {
    for (j=0; j<8; j++)
    {
        textattr(i + (j << 4));
        clrscr();
        cprintf("Acesta este un test\r\n");
        getch();
    }
   }
   clrscr();
}
```

{TEXTINFO.C - Obtine informatii despre modul text curent}

```
#include <conio.h>

int main(void)
{
   struct text_info ti;
   gettextinfo(&ti);
   clrscr();
```

```
   cprintf("stanga            %2d\r\n",ti.winleft);
   cprintf("sus               %2d\r\n",ti.wintop);
   cprintf("dreapta           %2d\r\n",ti.winright);
   cprintf("jos               %2d\r\n",ti.winbottom);
   cprintf("atribut           %2d\r\n",ti.attribute);
   cprintf("mod curent        %2d\r\n",ti.currmode);
   cprintf("inaltime
%2d\r\n",ti.screenheight);
   cprintf("latime
%2d\r\n",ti.screenwidth);
   cprintf("x curent          %2d\r\n",ti.curx);
   cprintf("y curent          %2d\r\n",ti.cury);
   getch();
}
```

{WINDOWS0.C - Creeaza si utilizeaza o fereastra in mod text}

```
#include <conio.h>
int i,j,a,b,l,c;
char ch;
struct text_info ti;
void fereastra(int li, int co, int st, int su)
{
    // desenare chenar
    window(st,su,st+co,su+li);
    textcolor(BLACK);textbackground(WHITE);
    cprintf("É");
    for (i=0; i<co-1; i++) cprintf("Í");
    cprintf("»");
```

```
    for (j=0; j<li-2; j++)
     {
     cprintf("ş");
     for (i=0; i<co-1; i++) cprintf(" ");
     cprintf("ş");
     }
    cprintf("Č");
    for (i=0; i<co-1; i++) cprintf("Í");
    cprintf("L");
    // stergere continut
    window(st+1,su+1,st+co-1,su+li-2);
    textcolor(WHITE);textbackground(BLACK);
    clrscr();
}
void main(void)
{
    // salvarea dimensiunilor ferestrei curente
    gettextinfo(&ti);
    // citirea parametrilor noii ferestre
    cprintf("Introduceti numarul de linii (min 3)
");
    scanf("%d", &l);  // nr. linii
    cprintf("Introduceti numarul de coloane (min 3)
");
    scanf("%d", &c);  // nr. coloane
    cprintf("Introduceti coordonata x stanga
");
    scanf("%d", &a);  // coloana stanga
    cprintf("Introduceti coordonata y sus
");
```

```
    scanf("%d", &b);   // linie sus
    clrscr();
    fereastra(l,c,a,b);    // crearea noii ferestre
    while ((ch != '.'))
     {
     ch = getch();
     putch(ch);
     }
    delay(500);
    // refacerea ferestrei initiale

window(ti.winleft,ti.wintop,ti.winright,ti.winbott
om);
    clrscr();
}
```

6.6. Desfăşurarea lucrării

a) Să se modurile de lucru cu monitorul video.

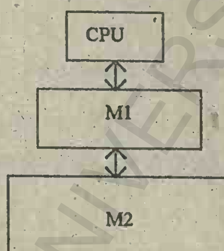
b) Să se studieze exemplele de la punctul 6.4. Să se verifice corectitudinea funcţionării programelor.

c) Să se studieze programul demostartiv BGIDEMO.C din kit-ul de instalare Borland C ++.

ANEXA : EXEMPLE DE PROBLEME

Problema 1

Se consideră un sistem de memorie ierarhizată , cu două nivele M_1 şi M_2, pentru un sistem de calcul , ca în figură :



Se notează :

C_1 , C_2 - costul per bit pentru memoriile de pe nivelul 1 , 2
S_1 , S_2 - capacitatea memoriilor de pe nivelul 1 , 2
t_1 , t_2 - timpii de acces pentru memoriile de pe nivelul 1 , 2

Funcţia de succes (rata de atingere) , H , este definită ca probabilitatea ca adresa logică generată de CPU să refere informaţie de pe nivelul M_1 .

a) Care este costul per bit , C , pentru întreg sistemul de memorie ierarhizată ?

b) În ce condiţii C se apropie de C_2 ?

c) Care este timpul mediu de acces, T , pentru întreg sistemul de memorie ?

d) Se notează cu $r = t_2 / t_1$ - raportul "vitezelor" şi cu $E = t_1 / T$ - eficienţa pentru accesarea sistemului de memorie ierarhizată .

Să se exprime E funcţie de r şi H.

Se cere reprezentarea grafică a lui $E = f(H)$ pentru r=1,2,10, şi 100.

e) Presupunînd că r=100 care este valoarea minimă a lui H pentru a avea $E > 0.90$?

Rezolvare

a) Costul memoriei pe nivelul 1 este C_1S_1
Costul memoriei pe nivelul 2 este C_2S_2
Costul total al memoriei este $C_1S_1 + C_2S_2$

Costul mediu este
$$C = \frac{C_1S_1 + C_2S_2}{S_1 + S_2}$$

b)
$$C = \frac{C_1}{1 + S_2 / S_1} + \frac{C_2}{1 + S_1 / S_2}$$

Dacă $S_2 >> S_1$ rezultă $1 + S_2 / S_1 >> 1$ şi $1 + S_1 / S_2 \approx 1$ deci $C_2 \approx C_1$

c) Timpul mediu de acces este :

$$T = \sum_{i=1}^{2} h_i T_i \text{ , cu } T_i = \sum_{k=1}^{i} t_k \text{ şi } h_i = H(s_i) - H(s_{i-1})$$

Rezultă $T_1 = t_1,\ T_2 = t_1 + t_2$

În continuare :

$$T = h_1T_1 + h_2T_2 = [\,H(s_1) - H(s_0)\,]\,T_1 + [\,H(s_2) - H(s_1)\,]\,T_2 =$$
$$= Ht_1 + (1 - H)(t_1 + t_2) = t_1 + t_2 - Ht_2$$

deoarece $H(s_2) = 1$ şi $H(s_0) = 0$.

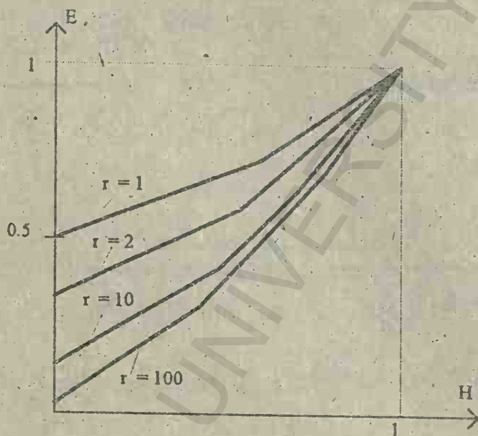
Se putea aplica direct formula $T = \sum_{i=1}^{2} F(s_{i-1})\, t_i = F(s_0)\, t_1 + F(s_1)\, t_2 =$

$= t_1 + (1 - H)\, t_2$

d) Se calculează $T / t_1 = 1 + r(1 - H)$ de unde rezultă :

$$E = \frac{1}{1 + r(1 - H)}$$

cu reprezentarea grafică :



e) $1 / (1 + r(1 - H)) = \alpha$ cu $\alpha = 0.9$ şi $r = 100$

Rezultă $H = H_{min} = 1 - (1/r)(1/\alpha - 1) = 1 - 10^{-3} / 0.9 \approx 1$

Problema 2

Fie secvenţa de pagini cerute într-un sistem de calcul cu un set de pagini rezidente s :

a b a c a b d b a c d

Să se indice numărul de excepţii (rata de excepţii) pentru algoritmii globali de reamplasare de tip : FIFO, LRU şi MRU (Most Recently Used) - opus algoritmului LRU pentru s = 2 şi s = 3.

Rezolvare

pentru s = 2

Secventa de pagini	a	b	a	c	a	b	d	b	a	c	d
Page Fault	*	*		*	*	*	*		*	*	*
Pagina fizică 1	a	a	a	c	c	b	b	b	a	a	d
Pagina fizică 2	-	b	b	b	a	a	d	d	d	c	c

Algoritmul FIFO - rata de excepţii 9 / 11

Secventa de pagini	a	b	a	c	a	b	d	b	a	c	d
Page Fault	*	*		*		*	*			*	*
Pagina fizică 1	a	a	a	a	a	a	a	a	a	a	a
Pagina fizică 2	-	b	b	c	c	b	d	b	b	c	d

Algoritmul LRU - rata de excepţii 8 / 11

Secventa de pagini	a	b	a	c	a	b	d	b	a	c	d
Page Fault	*	*		*	*	*	*	*	*	*	
Pagina fizică 1	a	a	a	c	c	c	c	b	a	c	c
Pagina fizică 2	-	b	b	b	a	b	d	d	d	d	d

Algoritmul MRU - rata de excepții 9 / 11

pentru s=3

Secventa de pagini	a	b	a	c	a	b	d	b	a	c	d
Page Fault	*	*		*			*		*		
Pagina fizică 1	a	a	a	a	a	a	d	d	d	d	d
Pagina fizică 2	-	b	b	b	b	b	b	b	a	a	a
Pagina fizică 3	-	-	-	c	c	c	c	c	c	c	c

Algoritmul FIFO - rata de excepții 5 / 11

Secventa de pagini	a	b	a	c	a	b	d	b	a	c	d
Page Fault	*	*		*			*			*	*
Pagina fizică 1	a	a	a	a	a	a	a	a	a	a	a
Pagina fizică 2	-	b	b	b	b	b	b	b	b	b	b
Pagina fizică 3	-	-	-	c	c	c	d	d	d	c	d

Algoritmul LRU - rata de excepţii 5 / 11

Secventa de pagini	a	b	a	c	a	b	d	b	a	c	d
Page Fault	*	*		*			*		*		
Pagina fizică 1	a	a	a	a	a	a	d	d	d	d	d
Pagina fizică 2	-	b	b	b	b	b	b	b	a	a	a
Pagina fizică 3	-	-	-	c	c	c	c	c	c	c	c

Algoritmul MRU - rata de excepţii 5 / 11

Discuţie

- Evident rata de excepţii scade cu cresterea lui s
- Rezultatele nu au grad de generalitate (prea puţine pagini cerute). Din această cauză diferenţele între diverşi algoritmi de reamplasare globali nu sînt sesizabile.

Problema 3

O structură pipe - line este caracterizată de tabela de rezervare :

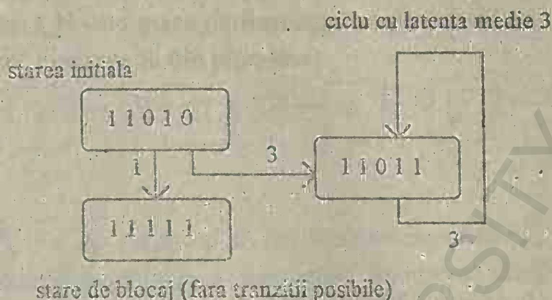
	t_0	t_1	t_2	t_3	t_4	t_5	t_6
S1	X					X	
S2			X				X
S3		X		X			
S4			X		X		

a) Să se determine setul interzis de latenţe , F
b) Să se determine vectorul de coliziune , C
c) Să se deseneze diagrama de stare

a) $F = \{ 2, 4, 5 \}$
b) $C = [C_5\, C_4\, C_3\, C_2\, C_1] = [1\ 1\ 0\ 1\ 0]$
c) Diagrama de stare :

Starea iniţială 11010 : tranziţii posibile 1 şi 3
Pentru tranziţia 1 noua stare este : (01101) OR (11010) = (11111), nu se mai pot face tranziţii (blocaj - apar coliziuni)

Pentru tranziţia 3 noua stare este : (00011) OR (11010) = (11011) - tranziţie posibilă 3 în starea (00011) OR (11010) = (11011) - (ciclu).



Procesele trebuie iniţiate cu latenţa 3.

Pentru iniţierea cu latenţa 1 se ajunge într-o stare de blocaj din care se poate ieşi prin iniţierea proceselor cu o latenţă mai mare decît 5.

Problema 4

Se consideră o structură pipe-line cu k nivele. Timpul de prelucrare pentru fiecare nivel este $\tau = 1$.

Fie p(N) probabilitatea ca instrucţiunea curentă la încărcare să depindă de intreucţiunea încărcată cu N cicli pipe-line mai înainte.

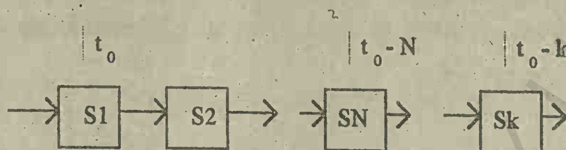
a) Să se calculeze timpul de execuţie , T , pentru M instrucţiuni.

b) Să se evalueze performanţa structurii pipe-line definită de relaţia :

$$P = \lim (M / T) , M \to \infty$$

Rezolvare

Fie structura pipe line din figură :



În cazul dependenţei instrucţiunii de la intrare de o instrucţiune aflată pe nivelul N , trebuie aşteptată încheierea execuţiei intrucţiunii care introduce dependenţă.

Pentru o singură instrucţiune timpul suplimentar introdus este (pentru $\tau = 1$):

$$(t_0 - N) - (t_0 - k) = k - N$$

Există p(N).M instrucţiuni "dependente" .

Timpul suplimentar total este : $T_s = p(N).M.(k - N)$

Timpul de execuţie normală (fără dependenţă) este :

$$T_n = k - 1 + M$$

Timpul total de execuţie este :

$$T = T_s + T_n = k - 1 + M + p(N).M.(k - N)$$

b) Performanţa structurii pipe-line este :

$$P = \frac{1}{1 + p(N).(k - N)}$$

Discuţie

Performanţa P depinde de p(N) , dar şi de localizarea dependenţei (nivelul N). Dacă N este mare performanţa creşte (instrucţiunea care produce dependenţă "iese" mai repede din pipe-line)

Problema 5

Se presupune că o operaţie scalară necesită un timp de execuţie de k ori mai mare decît o operaţie vectorială.

Se dă un program scris în limbaj scalar (cu operaţii scalare).

a) Care este procentajul din acest program ce trebuie rescris (cu operaţii vectoriale) pentru a se obţine o creştere de viteză de m ori ?

b) Să se repete punctul a) în situaţia în care q% din programul scalar **nu** poate fi rescris cu operaţii vectoriale .

Rezolvare

a) Se notează :

n - numărul de opearţii ale programului original

s - numărul de operaţii ce rămîn scalare (din numărul de operaţii ce pot fi rescrise)

v - numărul de operaţii ce vor fi "vectorizate" ,

t_s - timpul de execuţie pentru o operaţie scalară

t_v - timpul de execuţie pentru o operaţie vectorială

Există relaţia : $t_s / t_v = k$

Numărul de operaţii rămîne constant : $n = s + v$

Timpul de execuţie al programului original este : $nt_s = (s + v)\, t_s$

Timpul de execuţie al programului vectorizat este : $st_s + vt_v$

Rezultă creşterea de viteză : $m = \dfrac{(s+v)t_s}{st_s + vt_v} = \dfrac{(s+v)k}{ks+v}$

Procentajul din program ce trebuie rescris (vectorizat) este $x = v/(s+v)$

Rezultă : $m = \dfrac{k}{k(1-x)+x}$ de unde :

$$x = \frac{k}{k-1} \cdot \frac{m-1}{m}$$

b) Se utilizează notaţiile de la a)

Numărul de operaţii ce nu se pot vectoriza este : nq

Se pot vectoriza un număr de operaţii egal cu : $(1-q)n = s+v$

Rezultă :

$n = (s+v)/(1-q)$ sau $n/(s+v) = 1/(1-q)$ sau

$nq + s = (s+v)/(1-q) - v$

Timpul de execuţie pentru programul vectorizat este : $nqt_s + st_s + vt_s$
Rezultă creşterea de viteză :

$$m = \frac{(s+v+nq)t_s}{nqt_s + st_s + vt_s} = \frac{k(s+v)[1+q/(1-q)]}{[(s+v)/(1-q) - v]k + v}$$

Cu notaţia $x = v/(s+v)$ - procentajul de operaţii ce trebuie rescrise din totalul operaţiilor ce pot fi vectorizate - introdusă la a) rezultă :

$$m = \frac{k[1+q/(1-q)]}{[1/(1-q)-x]k+x}$$

Rezultă

$$x = \frac{1}{1-q} \cdot \frac{k}{k-1} \cdot \frac{m-1}{m}$$

Discuţie

Tabelul următor exemplifică relaţiile obţinute (k = 10):

q = 0%		q = 15 %	
m = 2	x = 55%	m = 2	x = 64%
m = 4	x = 83%	m = 4	x = 97%
m = 6	x = 92%	m = 6	Nu (x>1)

8.300 Lei